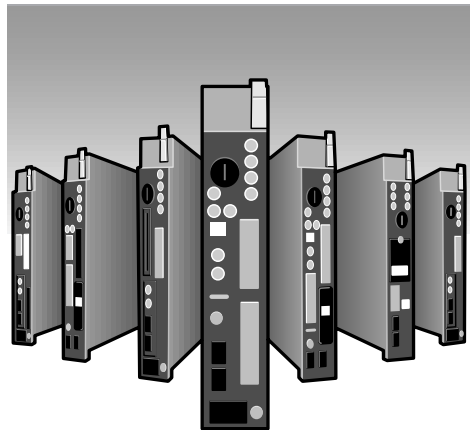




Allen-Bradley

***PLC-5
Programmable
Controllers***

Instruction Set Reference



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) 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 the Allen-Bradley Company 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, the Allen-Bradley Company cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Allen-Bradley Company 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 the Allen-Bradley Company is prohibited.

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



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 the hazard
- recognize the consequences

Important: Identifies information that is especially important for successful application and understanding of the product.

Ethernet is a registered trademark of Intel Corporation, Xerox Corporation, and Digital Equipment Corporation.

Data Highway Plus, DH+, PLC, PLC-5, PLC-5/11, -5/20, -5/26, -5/30, -5/40, -5/46, -5/40L, -5/60, -5/60L, -5/80, -5/86, -5/20E, -5/40E, and -5/80E are trademarks of Rockwell Automation.

Allen-Bradley is a trademark of Rockwell Automation, a core business of Rockwell International Corporation.

PLC-5 Instruction Set Alphabetical Listing

For this Instruction:	See Page:	For this Instruction:	See Page:	For this Instruction:	See Page:	For this Instruction:	See Page:
ABL	17-5 ¹	CMP	3-3	JSR	13-12	RES	2-25
ACB	17-7 ¹	COP	9-20	LBL	13-5	RET	13-12
ACI	17-9 ¹	COS	4-21 ¹	LEQ	3-9	RTO	2-13
ACN	17-10 ¹	CPT	4-5	LES	3-10	SBR	13-12
ACS	4-13 ¹	CTD	2-20	LFL	11-5 ¹	SDS	18-2
ADD	4-14	CTU	2-18	LFU	11-5 ¹	SFR	13-23 ¹
AEX	17-11 ¹	DDT	10-2	LIM	3-11	SIN	4-27 ¹
AFI	13-19	DEG	6-5 ¹	LN	4-23 ¹	SQI	12-2
AHL	17-12 ¹	DFA	18-3	LOG	4-24 ¹	SQL	12-2
AIC	17-14 ¹	DIV	4-22	MCR	13-3	SQO	12-2
AND	5-2	DTR	10-8	MEQ	3-13	SQR	4-28
ARD	17-15 ¹	EOT	13-24	MOV	7-4	SRT	4-29 ¹
ARL	17-18 ¹	EQU	3-6	MSG	16-2	STD	4-31 ¹
ASC	17-21 ¹	FAL	9-2	MUL	4-25	SUB	4-34
ASN	4-15 ¹	FBC	10-2	MVM	7-5	TAN	4-35 ¹
ASR	17-22 ¹	FFL	11-5	NEG	4-26	TND	13-19
ATN	4-16 ¹	FFU	11-5	NEQ	3-15	TOD	6-3
AVE	4-17 ¹	FLL	9-21	NOT	5-4	TOF	2-9
AWA	17-23 ¹	FOR	13-8	NXT	13-8	TON	2-5
AWT	17-26 ¹	FRD	6-4	ONS	13-20	UID	13-25 ¹
BRK	13-8	FSC	9-15	OR	5-6	UIE	13-26 ¹
BSL	11-2	GEQ	3-7	OSF	13-22 ¹	XIC	1-3
BSR	11-2	GRT	3-8	OSR	13-21 ¹	XIO	1-4
BTD	7-2	IDI	1-10 ²	OTE	1-5	XOR	5-8
BTR	15-4	IDO	1-11 ²	OTL	1-6	XPY	4-36 ¹
BTW	15-4	IIN	1-8	OTU	1-7		
CIO	15-25 ²	IOT	1-9	PID	NO TAG		
CLR	4-20	JMP	13-5	RAD	6-6 ¹		

¹ Enhanced PLC -5 processors only.

² 6200 programming software with ControlNet PLC-5 processors only

See Table A for guidelines on choosing the appropriate instruction for the operation you want to perform. Table B lists some examples.

Table A
Choosing an Instruction Category

If You Want to Perform this Operation:	Use this Instruction Category:	
examine, check or control	2-state device or condition multiple 2-state devices or conditions	bit level multi-bit
move, copy, change, compute, compare	analog values, codes multiple sets of values	element level file instructions
convert	conversion instructions	
time or delay	timer	
count	counter	
shift or track	bit shift	
sequence	sequencer	
PID	PID	
message sending/receiving	message	
transfer data to/from modules	block transfer or ControlNet transfer	
diagnostics, fault handling	diagnostics	
control the flow of your program	program control	

Table B
Example Operations

If Your Application Calls for Operations such as:	Use:
detecting when a limit switch closes	bit level
changing the temperature preset	element level
transfer analog data	block transfer
turn on a motor 10 seconds after a pump is activated	timing
move 1 of 3 recipes into a work area	multi-element
keep track of parts as they move from station to station	shifting
keep track of total parts in a bin	counting

Summary of Changes

New Information Added to this Manual

The list below summarizes the changes that have been made to this manual since the last printing:

For this Update Information:	See Chapter:
Converting non-decimal numbers with the FRD instruction	6
How non-existing, indirect addresses affect the COP and FLL instructions	9
How the .POS value operates in sequencer instructions	12
Using a RET instruction	13
Using the PID bias term	14
Using the no zero crossing (.NOZC) and no back calculation (.NOBC) features in the PD control block	14
Clarification to error code 89 for MSG instruction	16
Ethernet PLC-5 processors now support SLC Typed Read and SLC Typed Write MSG instructions	16
Configuring a multihop MSG instruction over Ethernet or over ControlNet	16
Monitoring the status of the .EN bit in a continuous MSG instruction	16

To help you find new information and updated information in this release of the manual, we have included change bars as shown to the left of this paragraph.

Summary of Changes

Notes:

Preface

Conventions

This manual uses the following conventions:

- Unless otherwise stated:

References to:	Include these Allen-Bradley Processors:
Classic PLC-5 processors	PLC-5/10™, -5/12™, -5/15™, -5/25™, and -5/VME™ processors.
Enhanced PLC-5 processors	PLC-5/11™, -5/20™, -5/30™, -5/40™, -5/40L™, -5/60™, -5/60L™, and -5/80™ processors. Note: Unless otherwise specified, Enhanced PLC-5 processors include Ethernet PLC-5, ControlNet PLC-5, Protected PLC-5 and VME PLC-5 processors.
Ethernet PLC-5 processors	PLC-5/20E™, -5/40E™, and -5/80E™ processors.
ControlNet PLC-5 processors	PLC-5/20C™, -5/40C™, -5/46C™, and -5/80C™ processors.
Protected PLC-5 processors ¹	PLC-5/26™, -5/46™, and -5/86™ processors.
VME PLC-5 processors	PLC-5/V30™, -5/V40™, -5/V40L™, and -5/V80™ processors. See the PLC-5/VME VMEbus Programmable Controllers User Manual for more information.

¹ Protected PLC-5 processors alone do not ensure PLC-5 system security. System security is a combination of the Protected PLC-5 processor, the software, and your application expertise.

- Words in square brackets represent actual keys that you press.
For example:
[Enter]; [F1] – Online Programming/Documentation
- Words that describe information that you have to provide are shown in italics. For example, if you have to type a file name, this is shown as:
filename
- Messages and prompts that the terminal displays are shown as:
Press a function key

Preface

Notes:

**Relay-Type Instructions
XIC, XIO, OTE, OTL, OTU, IIN, IOT,
IDI, IDO**

Chapter 1
Using Relay-Type Instructions 1-1
 I/O Image Files in Data Storage 1-2
 Rung Logic 1-2
Examine On (XIC) 1-3
Examine Off (XIO) 1-3
Energize (OTE) 1-4
Latch (OTL) 1-4
Unlatch (OTU) 1-5
Immediate Input (IIN) 1-6
Immediate Output (IOT) 1-7
Immediate Data Input (IDI) 1-8
Immediate Data Output (IDO) 1-8
Using IDI and IDO Instructions 1-9

**Timer Instructions TON, TOF,
RTO Counter Instructions CTU,
CTD Reset RES**

Chapter 2
Using Timers and Counters 2-1
 Using Timers 2-1
Entering Parameters 2-2
Timer Accuracy 2-3
Timer On Delay (TON) 2-4
 Using Status Bits 2-4
Timer Off Delay (TOF) 2-7
 Using Status Bits 2-7
Retentive Timer On (RTO) 2-10
 Using Status Bits 2-10
Using Counters 2-13
 Entering Parameters 2-13
Count Up (CTU) 2-15
 Using Status Bits 2-15
Count Down (CTD) 2-17
 Using Status Bits 2-17
Timer and Counter Reset (RES) 2-20

Compare Instructions

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

Chapter 3

Using Compare Instructions	3-1
Using Arithmetic Status Flags	3-2
Compare (CMP)	3-2
Entering the CMP Expression	3-2
Determining the Length of an Expression	3-3
Equal to (EQU)	3-5
Greater than or Equal to (GEQ)	3-5
Greater than (GRT)	3-6
Less than or Equal to (LEQ)	3-6
Less than (LES)	3-7
Limit Test (LIM)	3-7
Entering Parameters	3-7
Mask Compare Equal to (MEQ)	3-9
Entering Parameters	3-9
Not Equal to (NEQ)	3-10

Compute Instructions

**CPT, ACS, ADD, ASN, ATN, AVE,
CLR, COS, DIV, LN, LOG, MUL, NEG,
SIN, SRT, SQR, STD, SUB, TAN, XPY**

Chapter 4

Using Compute Instructions	4-1
Using Arithmetic Status Flags	4-2
Data Types and the Compute Instruction	4-3
Using Floating Point Data Types	4-4
Compute (CPT)	4-5
Entering the CPT Expression	4-5
Determining the Length of an Expression	4-7
Determining the Order of Operation	4-8
Expression Examples	4-8
Entering the Destination	4-9
Using CPT Functions	4-9
Arc Cosine (ACS)	4-11
Addition (ADD)	4-12
Arc Sine (ASN)	4-13
Arc Tangent (ATN)	4-14
Average File (AVE)	4-15
Entering Parameters	4-15
Using Status Bits	4-16
Clear (CLR)	4-17
Cosine (COS)	4-18
Divide (DIV)	4-19
Natural Log (LN)	4-20
Log to the Base 10 (LOG)	4-21
Multiply (MUL)	4-22
Negate (NEG)	4-23
Sine (SIN)	4-24
Square Root (SQR)	4-25

	Sort File (SRT)	4-26
	Entering Parameters	4-26
	Using Status Bits	4-27
	Standard Deviation (STD)	4-28
	Entering Parameters	4-29
	Using Status Bits	4-29
	Subtract (SUB)	4-31
	Tangent (TAN)	4-32
	X to the Power of Y (XPY)	4-33
Logical Instructions	Chapter 5	
AND, NOT, OR, XOR	Using Logical Instructions	5-1
	Using Arithmetic Status Flags	5-1
	AND Operation (AND)	5-2
	NOT Operation (NOT)	5-3
	OR Operation (OR)	5-4
	Exclusive OR Operation (XOR)	5-5
Conversion Instructions	Chapter 6	
FRD and TOD, DEG and RAD	Using the Conversion Instructions	6-1
	Using Arithmetic Status Flags	6-1
	Convert to BCD (TOD)	6-2
	Convert from BCD (FRD)	6-2
	Degree (DEG)	
	(Enhanced PLC-5 Processors Only)	6-3
	Radian (RAD)	
	(Enhanced PLC-5 Processors Only)	6-4
Bit Modify and Move Instructions	Chapter 7	
BTD, MOV, MVM	Using Bit Modify and Move Instructions	7-1
	Bit Distribute (BTD)	7-2
	Entering Parameters	7-2
	Move (MOV)	7-3
	Masked Move (MVM)	7-4
	Entering Parameters	7-4
File Instruction Concepts	Chapter 8	
	Concepts of File Operation	8-1
	Entering Parameters	8-1
	Using the Control Structure	8-2
	Manipulating File Data	8-3
	Choosing Modes of Block Operation	8-5
	All Mode	8-5
	Numerical Mode	8-6
	Incremental Mode	8-7
	Special Case, Numerical Mode with Words Per Scan = 1	8-8

File Instructions

FAL, FSC, COP, FLL

Chapter 9	
Using File Instructions	9-1
File Arithmetic and Logic (FAL)	9-2
Using Status Bits	9-4
FAL Copy Operations	9-5
FAL Arithmetic Operations	9-7
Upper and Lower Limits	9-7
FAL Logic Operations	9-12
FAL Convert Operations	9-14
File Search and Compare (FSC)	9-14
Using Status Bits	9-15
FSC Search and Compare Operations	9-17
Data Conversion	9-17
File Search Operation	9-17
File Copy (COP)	9-19
Entering Parameters	9-19
File Fill (FLL)	9-20
Entering Parameters	9-20

Diagnostic Instructions

FBC, DDT, DTR

Chapter 10	
Using Diagnostic Instructions	10-1
File Bit Comparison (FBC) and Diagnostic Detect (DDT)	10-2
Selecting the Search Mode	10-2
One Mismatch at a Time	10-2
All Per Scan	10-3
Entering Parameters	10-4
Using Status Bits	10-5
Data Transitional (DTR)	10-8
Entering Parameters	10-8

Shift Register Instructions

BSL, BSR, FFL, FFU, LFL, LFU

Chapter 11	
Applying Shift Registers	11-1
Using Bit Shift Instructions	11-2
Entering Parameters	11-2
Using Status Bits	11-3
Using FIFO and LIFO Instructions	11-5
Entering Parameters	11-5
Using Status Bits	11-6

Sequencer Instructions

SQO, SQI, SQL

Chapter 12	
Applying Sequencers	12-1
Using Sequencer Instructions	12-2
Entering Parameters	12-2
Using Status Bits	12-4
Resetting the Position of SQO	12-6
Using SQI Without SQO	12-7

**Program Control Instructions MCR,
JMP, LBL, FOR, NXT, BRK, JSR,
SBR, RET, TND, AFI, ONS, OSR, OSF,
SFR, EOT, UIE, UID**

Chapter 13

Selecting Program Flow Instructions	13-1
Master Control Reset (MCR)	13-2
Jump (JMP) and Label (LBL)	13-3
Using JMP	13-4
Using LBL	13-4
For Next Loop (FOR, NXT), Break (BRK)	13-5
Entering Parameters	13-6
Using FOR	13-6
Using BRK	13-7
Using NXT	13-7
Jump to Subroutine (JSR), Subroutine (SBR), and Return (RET)	13-8
Passing Parameters	13-8
Entering Parameters	13-10
Nesting Subroutine Files	13-10
Using JSR	13-11
Using SBR	13-11
Using RET	13-12
Temporary End (TND)	13-13
Always False (AFI)	13-13
One Shot (ONS)	13-14
One Shot Rising (OSR)	13-15
Entering Parameters	13-15
One Shot Falling (OSF)	13-16
Entering Parameters	13-16
Sequential Function Chart Reset (SFR)	13-17
Entering Parameters	13-17
End of Transition (EOT)	13-18
User Interrupt Disable (UID)	13-19
User Interrupt Enable (UIE)	13-20

Process Control Instruction PID

Chapter 14

Using PID	14-1
PID Features	14-2
Using PID Equations	14-2
Conversion of Gain Constants	14-3
Integral Term Implementation	14-3
Derivative Term	14-4
Setting Input/Output Ranges	14-5
Implementing Scaling to Engineering Units	14-5
Setting the Dead Band	14-6
Using Zero-Crossing	14-6
Using No Zero Crossing	14-7
Selecting the Derivative Term (Acts on PV or Error)	14-7

Setting Output Alarms	14-7
Using Output Limiting	14-7
Anti-Reset Windup	14-8
Using a Manual Mode Operation (Bumpless Transfer)	14-8
Set Output	14-8
Feedforward or Output Biasing	14-9
Resume Last State	14-9
PID Instruction	14-10
Using No Back Calculation	14-11
Operational Status Bits	14-11
Integer Block	14-11
PD Block	14-12
Entering Parameters	14-12
Using an Integer Data File Type for the Control Block	14-14
Using Control Block Values	14-16
Using a PD File Type for the Control Block	14-18
Using Control Block Values	14-23
Programming Considerations	14-25
Run Time Errors	14-25
Transferring Data to the PID Instruction	14-25
Loop Considerations	14-26
Number of PID Loops	14-26
Loop Update Time	14-26
Decaling Inputs	14-27
PID Examples	14-29
Integer Block (N) Examples	14-29
Main Program File	14-29
STI Program File	14-30
RTS Program File	14-32
PD Block Examples	14-33
Main Program File	14-33
STI Program File	14-34
RTS Program File	14-36
Ladder Logic Simulation of a Manual Control Station	14-37
Cascading Loops	14-38
Ratio Control	14-38
Process Variable Tracking	14-39
PID Theory	14-40

Block-Transfer Instructions
BTR and BTW and ControlNet I/O
Transfer Instruction CIO

Chapter 15

Using Block Transfer and ControlNet I/O	
Transfer Instructions	15-1
Using Block Transfer Instructions	15-1
Block-Transfer Read (BTR) and Block-Transfer Write (BTW).	15-3
Block-Transfer Request Queue	15-3
Entering Parameters	15-4
Using Status Bits	15-6
Using the Control Block	15-8
Requested Word Count (.RLEN)	15-8
Transmitted Word Count (.DLEN)	15-8
File Number (.FILE)	15-9
Element Number (.ELEM)	15-9
Selecting Continuous Operation	15-10
Selecting Non-Continuous Operation	15-12
Block Transfer Timing – Classic PLC-5 Processors	15-13
Instruction Run Time	15-13
Waiting Time in the Queue	15-13
Transfer Time	15-13
Block Transfer Timing – Enhanced PLC-5 Processors	15-14
Instruction Run Time	15-14
Waiting Time in the Holding Area	15-14
Transfer Time	15-14
Programming Examples	15-15
Example Bidirectional Alternating Block-Transfer	15-16
Example Bidirectional Alternating Repeating Block-Transfer	15-17
Example Bidirectional Continuous Block-Transfer	15-18
Example Directional Non-Continuous Block-Transfer	15-19
Example Directional Repeating Block Transfer	15-19
Example Directional Continuous Block-Transfer	15-20
Example Buffering Block Transfer-Data	15-21
ControlNet I/O Transfer (CIO) Instruction	15-22
Control Block Address	15-22
Using the CIO Instruction	15-23
Using Status Bits	15-24
Using the CT Control Block	15-25

Message Instruction MSG**Chapter 16**

Using the Message Instruction	16-1
Message (MSG)	16-1
Entering Parameters	16-2
Control Block Address	16-2
MSG Data Entry Screen	16-3
Using the Message Instruction for Ethernet Communications	16-5
Entering Parameters	16-5
Using the Message Instruction for PLC-5 Ethernet Interface Module Communications	16-7
Entering Parameters	16-7
Configuring an Ethernet Multihop MSG Instruction	16-9
Using the Message Instruction for ControlNet Communications	16-10
Control Block Address	16-10
Configuring a ControlNet Multihop MSG Instruction	16-11
Using Status Bits	16-12
Using the Control Block	16-13
Error Code (.ERR)	16-13
Requested Length (.RLEN)	16-13
Transmitted Length (.DLEN)	16-13
Entering Parameters	16-14
Communication Command	16-14
External Data Table Addresses	16-15
PLC-2 to PLC-5 Compatibility Files	16-15
Sending SLC Typed Logical Read and Typed Logical Write Commands	16-16
Monitoring a Message Instruction	16-17
Selecting Continuous Operation	16-18
Selecting Non-Continuous Operation	16-19
MSG Timing	16-20
Error Codes	16-22

ASCII Instructions

**ABL, ACB, ACI, ACN, AEX, AIC, AHL,
ARD, ARL, ASC, ASR, AWA, AWT**

Chapter 17

Using ASCII Instructions	
Enhanced PLC-5 Processors Only	17-1
Using Status Bits	17-2
Using the Control Block	17-3
Length (.LEN)	17-3
Position (.POS)	17-3
Using Strings	17-3
Test Buffer for Line (ABL)	17-4
Entering Parameters	17-4
Number of Characters in Buffer (ACB)	17-5
Entering Parameters	17-5
ASCII String to Integer (ACI)	17-6
ASCII String Concatenate (ACN)	17-7
ASCII String Extract (AEX)	17-7
Entering Parameters	17-7
ASCII Set or Reset Handshake Lines (AHL)	17-8
Entering Parameters	17-8
ASCII Integer to String (AIC)	17-9
ASCII Read Characters (ARD)	17-10
Entering Parameters	17-10
ASCII Read Line (ARL)	17-12
Entering Parameters	17-12
ASCII String Search (ASC)	17-14
Entering Parameters	17-14
ASCII String Compare (ASR)	17-15
ASCII Write with Append (AWA)	17-15
Entering Parameters	17-15
ASCII Write (AWT)	17-17
Entering Parameters	17-17

**Custom Application Routine
Instructions SDS, DFA**

Chapter 18

Chapter Objectives	18-1
Smart Directed Sequencer (SDS) Overview	18-2
Programming the SDS Instruction	18-2
Diagnostic Fault Annunciator (DFA) Overview	18-3
Programming the DFA Instruction	18-3

Instruction Timing and Memory Requirements

Appendix A-1

Instruction Timing and Memory Requirements.	A-1
Timing for Enhanced PLC-5 Processors.	A-2
Bit and Word Instructions.	A-2
File Instructions.	A-5
Timing for Classic PLC-5 Processors.	A-10
Bit and Word Instructions.	A-10
File Instructions.	A-13
Program Constants	A-17
Direct and Indirect Elements: Enhanced PLC-5 Processors	A-17
Direct and Indirect Elements: Classic PLC-5 Processors	A-18
Indirect Bit or Elements Addresses: Classic PLC-5 Processors	A-19
Additional Timing Considerations: Classic PLC-5 Processors	A-20

SFC Reference

Appendix B-1

Appendix Objectives	B-1
SFC Status Information in the Processor Status File.	B-1
Memory Allocation	B-3
Dynamic Constraints – Classic PLC-5 Processors Only	B-5
Scanning Sequences.	B-7
Step and Transition Scanning	B-7
Selected Branch Scanning.	B-8
Simultaneous Branch Scanning.	B-9
SFC Example and Scan Sequence	B-11
Run Times – Classic PLC-5 Processors	B-12
Using Sequence Diagrams to Determine Run Time	B-13
Using Equations to Determine Run Time	B-14

Valid Data Types for Instruction Operands

Appendix C-1

Appendix Objectives	C-1
Instruction Operands and Valid Data Types	C-1

Relay-Type Instructions XIC, XIO, OTE, OTL, OTU, IIN, IOT, IDI, IDO

Using Relay-Type Instructions

Use relay-type instructions to monitor and control the status of bits in the data table, such as input bits or timer control-word bits. The relay instructions let you:

If You Want to:	Use this Instruction:	Found on Page:
Examine a bit for an ON condition	XIC	1-3
Examine a bit for an OFF condition	XIO	1-3
Hold a bit ON or OFF (non-retentive)	OTE	1-4
Latch a bit to ON (retentive)	OTL	1-4
Unlatch a bit to OFF (retentive)	OTU	1-5
Immediately update input image bits	IIN	1-6
Immediately update outputs	IOT	1-7
Immediately perform an update of the ControlNet™ data input file from the ControlNet memory buffers.	IDI	1-8
Immediately perform an update of the ControlNet memory buffers from the source file before the next output-image update	IDO	1-8

With these instructions, you can address bits in all sections of data storage, but the examples in this chapter only show how to address bits in the I/O image files.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

If you use relay-type instruction (OTE, OTL, or OTU) with indirect addresses to set or reset a bit in the control file of a block-transfer or message instruction, there may be conflicting results. Even though the bit instruction is executed to set or reset a bit, its result might be overwritten by the block-transfer or message operation that sets or resets the same bit. These are asynchronous operations. The last operation to set or reset the bit is the value that is saved in the data table.

I/O Image Files in Data Storage

The input image file in the processor stores the status of input sensors connected to input module terminals.

If the Input Sensor Is:	Then Its Corresponding Input Image Bit Is:
closed (on)	on (1)
open (off)	off (0)

You program instructions in ladder logic to monitor bits. Use a logical address for the bit.

The output image file controls the status of actuators wired to output module terminals.

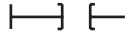
If the Output Image Bit Is:	Then Its Corresponding Output Is:
on (1)	energized (on)
off (0)	de-energized (off)

You program instructions in ladder logic to control bits.

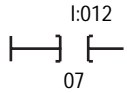
Rung Logic

As each conditioning instruction is executed, the addressed bit is examined to see if it matches a certain condition (on or off). If a complete path of true conditions examined for are found, the rung is set to true. The rung must contain a continuous path of true instructions from the start of the rung to the output for the output to be enabled.

Examine On (XIC)



Example:



If you find an ON condition at bit I:012/07 in the input table, set this instruction true. This bit corresponds to input terminal 7 of a module in I/O group 2 of I/O rack 1. If the input circuit is true, the instruction is true.

Description:

When a device closes its circuit, the module whose input terminal is wired to the device detects the closed circuit. The processor reflects this ON state in the data table. When the processor finds an XIC instruction that addresses the bit that corresponds to the input terminal, the processor determines whether the device is ON (closed). If the processor finds an ON state, it sets the logic for this instruction true; if the processor finds an OFF state, it sets the logic for the instruction not true.

If the XIC instruction is the only conditioning instruction on the rung, the processor enables the output instruction when the XIC instruction is true (input closed). The processor disables an output instruction when the XIC instruction is false (input open).

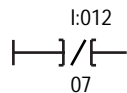
The examine-on instruction is true or false depending on whether the processor finds an ON or OFF condition at the addressed bit.

If the Bit Is:	Then the Instruction Is:	Bit Logic State:
on	true	1
off	false	0

Examine Off (XIO)



Example:



If you find an OFF condition at bit I:012/07 in the input table, set this instruction true. This bit corresponds to input terminal 7 of a module in I/O group 2 of I/O rack 1. If the input circuit is false, the instruction is true.

Description:

When a device opens its circuit, the module whose input terminal is wired to the device detects an open circuit. The processor reflects this OFF state in the data table. When the processor finds an XIO instruction that addresses the bit that corresponds to the input terminal, the processor determines whether the device is OFF (open). If the processor finds an OFF state, it sets the logic for this instruction true. If the processor finds an ON state, it sets the XIO instruction to false.

If the XIO instruction is the only conditioning instruction on the rung, the processor enables the output instruction when the XIO instruction is true (input open).

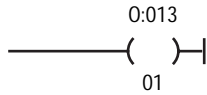
The examine off instruction is true or false depending on whether the processor finds an OFF or ON condition at the addressed bit.

If the Bit Is:	Then the Instruction Is:	Bit Logic State:
off	true	0
on	false	1

Energize (OTE)



Example:



Turn ON bit 0:013/01 of the output image table if the rung is true. Turn it OFF if the rung is false. This bit corresponds to output terminal 01 of a module in I/O group 3 of I/O rack 1.

Description:

Use the OTE instruction to control a bit in memory. If the bit corresponds to an output module terminal, the device wired to this terminal is energized when the instruction is enabled and de-energized when the instruction is disabled. If the input conditions that precede the OTE instruction are true, the processor enables the OTE instruction. If the input conditions that precede the OTE instruction are false, the processor disables the OTE instruction. When rung conditions become false, the corresponding device de-energizes.

An OTE instruction is similar to a relay coil. The OTE instruction is controlled by preceding input instructions; the relay coil is controlled by contacts in its hard-wired rung.

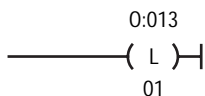
The OTE instruction tells the processor to control the addressed bit based on the rung condition:

If the Rung Is:	Then the Processor Turns the Bit:	Bit Logic State:
true	on	1
false	off	0

Latch (OTL)



Example:



Turn ON bit 0:013/01 of the output image table if the rung is true. This bit corresponds to output terminal 1 of a module in I/O group 3 of I/O rack 1.

Description:

The OTL instruction is a retentive output instruction that can only turn on a bit (it cannot turn off a bit). This instruction is usually used in pairs with an OTU (unlatch) instruction, with both instructions addressing the same bit.

When you assign an address to an OTL instruction that corresponds to a terminal of an output module, the output device wired to this terminal is energized when the processor sets (enables) the bit in processor memory. If the input conditions that precede the OTL instruction are true, the processor enables the OTL instruction. When rung conditions become false (after being true), the bit remains set and the corresponding output device remains energized. Use the OTU instruction to turn OFF the bit you latched on with the OTL instruction.

When enabled, the latch instruction tells the processor to turn on the addressed bit. Thereafter, the bit remains on, regardless of the rung condition, until the bit is turned off, typically by an unlatch (OTU) instruction in another rung.

If the Rung Is:	Then the Processor Turns the Bit:
true	on
false	no change

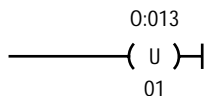
When the processor changes from Run to Program mode or when the processor loses power (and there is battery backup), the last true OTL instruction continues to control the bit in memory. The latched output device is energized even though the rung conditions that control the instruction may have gone false.

Important: The OTL instruction is retentive. When the processor loses power, is switched to Program mode or Test mode, or detects a major fault, outputs go off; but the states of retentive outputs are retained in memory. When the processor resumes operation in Run mode, retentive outputs immediately return to their previous states. Non-retentive outputs, such as OTE outputs, are reset.

Unlatch (OTU)



Example:



Turn OFF bit 0:013/01 of the output image table if the rung is true.
 This bit corresponds to output terminal 1 of a module in I/O group 3 in I/O rack 1.

Description:

The OTU instruction is a retentive output instruction that can only turn off a bit (it cannot turn on a bit). This instruction is usually used in pairs with an OTL (output latch) instruction, with both instructions addressing the same bit. The OTU instruction turns OFF the bit, which was turned ON (latched) by the OTL instruction.

When the processor changes from Run to Program mode or when the processor loses power (and there is battery backup), the bit is retained in the state set by the last rung of the latch/unlatch pair that was true.

The unlatch instruction tells the processor to turn off the addressed bit based on the rung condition. Thereafter, the bit remains off, regardless of the rung condition, until it is turned on, typically by a OTL instruction in another rung.

If the Rung is:	Then the Processor Turns the Bit:
true	off
false	no change

Immediate Input (IIN)

—————(IIN)—————

Example:

RRG
—————(IIN)—————

Where:

RR = I/O rack number
 00-03 PLC-5/10, -5/11, -5/12, -5/15, -5/20
 00-07 PLC-5/25, -5/30
 000-177 PLC-5/40, -5/40L
 000-277 PLC-5/60, -5/60L, -5/80
 G = I/O group number (0 - 7)

001
—————(IIN)—————

When the input conditions are true, update the input image word corresponding to I/O rack 0, group 1.

Description: The IIN instruction is an output instruction that, when enabled, updates a word of input-image bits before the next regular input-image update.

For inputs in the **local chassis**, the program scan is interrupted while the inputs of the addressed I/O group are examined. This sets the input-image bits to the current states of the inputs before the program scan continues. If the program reaches an enabled IIN instruction while a block-transfer with the local chassis is in progress, the processor completes the block-transfer before executing the IIN instruction.

For inputs in a **remote chassis**, the program scan is interrupted only to update the input image with the latest states of the inputs as found in the remote I/O buffer (from the most recent remote I/O scan). The inputs are not scanned before the program scan continues.

Place the rung with the IIN instruction immediately before rungs that examine critical input bits updated by the IIN instruction.

For the IIN instruction, you only need to enter the I/O rack number and the I/O group number; you do not enter a file number.

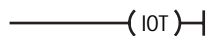


ATTENTION: Do not enter an address that includes a file number, such as I:027. The processor interprets the bit pattern found at that address as the I/O rack and I/O group number of the inputs to update. Unexpected operation will result with possible damage to equipment and injury to personnel.

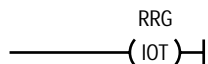
For more information on I/O scanning and block-transfers, see chapter 15.

Immediate Output (IOT)

Description: The IOT instruction is an output instruction that, when enabled, updates an I/O group of outputs before the next normal output-image update.

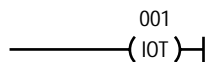


Example:



Where:

- RR = I/O rack number
00-03 PLC-5/10, -5/11, -5/12, -5/15, -5/20
00-07 PLC-5/25, -5/30
000-177 PLC-5/40, -5/40L
000-277 PLC-5/60, -5/60L, -5/80
- G = I/O group number (0 - 7)



When the input conditions are true, update the output image word corresponding to I/O rack 0, group 1.

For outputs in the **local chassis**, the program scan is interrupted while the outputs of the addressed I/O group are examined. This sets the output circuits to the current states of the output bits in the output image table before the program scan continues. If the program reaches an enabled IOT instruction while a block-transfer is in progress, the processor completes the block-transfer before executing the IOT instruction.

For outputs in a **remote chassis**, the program scan is interrupted only to update the remote I/O buffer with the current states of the output-image bits. This makes these states immediately available for the next remote I/O scan while the program scan continues. The outputs are not scanned before the program scan continues.

Place the rung with the IOT output instruction immediately after rungs that control critical output image bits to be updated by the IOT instruction.

For the IOT instruction, you only need to enter the I/O rack number and the I/O group number; you do not need to enter the file number.



ATTENTION: Do not enter an address that includes a file number, such as O:027. The processor interprets the bit pattern found at that address as the I/O rack and I/O group number of the outputs to be updated. Unexpected operation will result with possible damage to equipment and injury to personnel.

For more information on I/O scanning and block-transfers, see chapter 15.

Immediate Data Input (IDI)

IDI	
IMMEDIATE DATA INPUT	
Data file offset	232
Length	10
Destination	N10:232

Description: When the rung goes true, the IDI instruction performs an immediate update of the ControlNet data input file from the ControlNet memory buffers before the next normal input-image update (which occurs at the end of the program scan).

To program an IDI instruction, you must provide the processor with the following information that it stores in its control block:

- **Data file offset** specifies the offset into the Data Input File (DIF) where words are read – can be an immediate value (0-999) or a logical address that specifies the data image file offset.
- **Length** specifies the number of words to be transferred – an immediate value (0-64) or a logical address that specifies the number of words to be transferred.
- **Destination** specifies a data table address to be used as the destination of the words to be transferred.

Important: The Destination should be the matching data-table address in the Data Input File (DIF) except when you use the instruction to ensure data-block integrity in the case of Selectable Timed Interrupts (STIs). For more information, see page 1-9.

Immediate Data Output (IDO)

IDO	
IMMEDIATE DATA OUTPUT	
Data file offset	232
Length	10
Source	N7:232

Description: When the rung goes true, the IDO instruction performs an immediate update of the ControlNet memory buffers from the source file before the next output-image update, sending the updated data output file information across the ControlNet network to the appropriate ControlNet device.

To program an IDO instruction, you must provide the processor with the following information that it stores in its control block:

- **Data file offset** specifies the offset into the Data Output File (DOF) where words are written – can be an immediate value (0-999) or a logical address that specifies the data image file offset.
- **Length** specifies the number of words to be transferred – an immediate value (0-64) or a logical address that specifies the number of words to be transferred.
- **Source** specifies a data table address to be used as the source of the words to be transferred.

Important: The Source should be the matching data-table address in the Data Output File (DOF) except when you use the instruction to ensure data-block integrity in the case of Selectable Timed Interrupts (STIs). For more information, see page 1-9.

Using IDI and IDO Instructions

You can use the IDI and IDO instructions for immediate data input and output on ControlNet.

For more detailed information about writing ladder programs, see your programming manual.

Important: Be careful when using Selectable Timed Interrupts (STIs) with a program on a ControlNet network.

A Selectable Timed Interrupt (STI) periodically interrupts primary program execution in order to run a subprogram to completion. If an STI occurs while a normal ControlNet non-discrete I/O transfer or a ControlNet Immediate Data I/O instruction (IDI or IDO) is in progress and they both operate on the same set of data, the integrity of that block of data is jeopardized.

To ensure data-block integrity, write your STI routine so that it operates on its own copy of the data block that it needs. Use ControlNet Immediate Data I/O instructions (IDI and IDO) within your STI to copy the needed block of data out to and back from a temporary location that is different from that used by the normal data table.

For detailed information on STIs, see your software user manual.

Notes:

Timer Instructions TON, TOF, RTO Counter Instructions CTU, CTD Reset RES

Using Timers and Counters

Timers and counters let you control operations based on time or number of events. Table 2.A lists the available timer and counter instructions.

Table 2.A
Available Timer and Counter Instructions

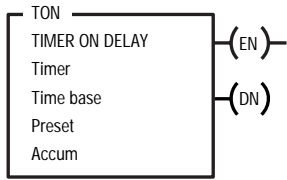
If You Want to:	Use this Instruction:	Found on Page:
Delay turning on an output	TON	2-4
Delay turning off an output	TOF	2-7
Time an event retentively	RTO	2-10
Count up	CTU	2-15
Count down	CTD	2-17
Reset a counter, timer, or counter instruction	RE	2-20

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Timers

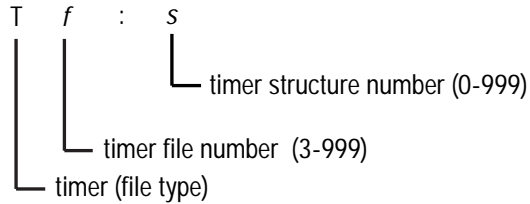
Before you program timer instructions, you need to understand the parameters that you enter for timer instructions and how timer accuracy works.

Entering Parameters



To program a timer instruction, provide the processor with the following information:

- **Timer** is the timer control address in the timer (T) area of data storage. Use the following address format:



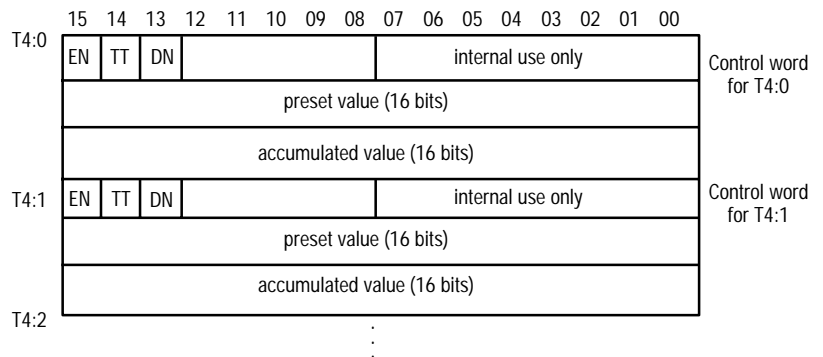
Important: You can use any timer file number from 3 to 999; however, the default timer file number is 4. If you want to specify a timer file number as any file between 3 and 8 (other than the default 4), you must first delete the entire default file for that number, and then create the timer file. For example, if you want a timer file number as file 3, you must first delete the entire default binary file and then create the timer file as file 3.

To access a timer status bit, preset, or accumulated value stored at the timer control address, use the following address format:

Status Bit	Preset	Accumulated Value
Tf:s.sb	Tf:s.PRE	Tf:s.ACC

The *sb* specifies a status bit mnemonic, such as .DN

Important: The processor stores timer status bits and the preset and accumulated values in a 48-bit storage structure (three 16-bit words) in a timer file (T).



- **Time Base** determines how the timer operates. Table 2.B lists the possible time bases.

Table 1.B
Available Time Base Values

Enter This Time Base:	The Accumulated Value Range Is:
1 second	to 32,767 time-base intervals (to 9.1 hours)
0.01 seconds (10ms)	to 32,767 time-base intervals (to 5.5 minutes)

- **Preset** specifies the value which the timer must reach before the processor sets the done bit (.DN). You must enter a preset value from 0-32,767. The processor stores the preset value as a 16-bit integer value.

Important: The Preset value operates differently if you are using a TOF instruction. See page 2-7 for more information.

- **Accumulated Value** is the number of time increments the instruction has counted. When enabled, the timer updates this value continually. Typically, enter zero when programming the instruction. If you enter a value, the instruction starts counting time base intervals from that value. If the timer is reset, the accumulated value is zero. The range for the accumulated value is 0-32,767. The processor stores the accumulated value as a 16-bit integer.

Important: The Accumulated value operates differently if you are using a TOF instruction. See page 2-7 for more information.

Timer Accuracy

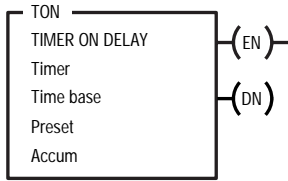
Timer accuracy refers to the length of time between the moment the processor enables a timer instruction and the moment the processor completes the timed interval. Timer accuracy depends on the processor clock tolerance and the time base. The clock tolerance is $\pm 0.02\%$. This means that a timer could time out early or late by 0.01 seconds (10ms) for a 0.01 second time base or 1 second for a 1 second time base.

The 0.01-second timer maintains accuracy with a program scan of up to 2.5 seconds; the 1-second timer maintains accuracy with a program scan of up to 1.5 seconds. If your programs can exceed 1.5 or 2.5 seconds, repeat the timer instruction rung so that the rung is scanned within these limits.

The displayed accumulated value of a timer shows actual time but is dependent on CRT update time. The accumulated value might appear to be less than the preset when the done bit is set.

Timer On Delay (TON)

Description:



Use the TON instruction to turn an output on or off after the timer has been on for a preset time interval. The TON instruction starts accumulating time when the rung goes true, and continues until one of the following happens:

- the accumulated value equals its preset value
- the rung goes false
- a reset instruction resets the timer
- the SFC step goes inactive
- the processor resets the accumulated value when the rung conditions go false, regardless of whether the timer timed out or not

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address status bits by mnemonic.

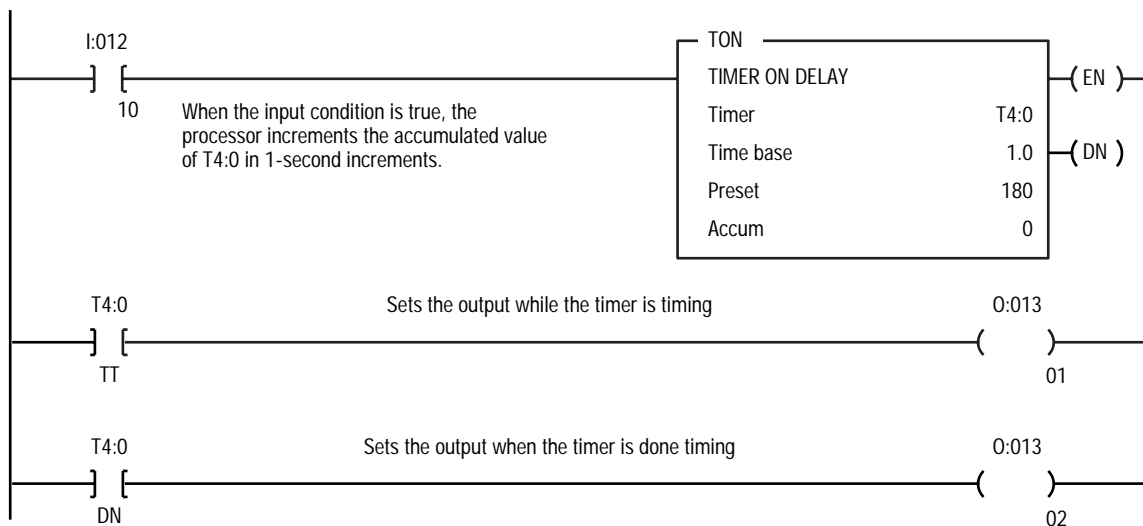
This Bit:	Is Set When:	Indicates:	And Remains Set Until One of the Following Occurs:
Timer Enable.EN (bit 15)	the rung goes true	that the timer is enabled	<ul style="list-style-type: none"> • the rung goes false • a reset instruction resets the timer • the SFC step goes inactive
Timer Timing Bit .TT (bit 14)	the rung goes true	that a timing operation is in progress	<ul style="list-style-type: none"> • the rung goes false • the .DN bit is set (.ACC = .PRE) • a reset instruction resets the timer • the associated SFC step goes inactive
Timer Done Bit .DN (bit 13)	the accumulated value is equal to the preset value	that a timing operation is complete	<ul style="list-style-type: none"> • the rung goes false • a reset instruction resets the timer • the associated SFC step goes inactive

If you set the done bit .DN using an OTE instruction, for example, you can pause the timer. The .EN and .TT bits remain set, but the accumulated value does not increment. Timing resumes when you clear the .DN bit. If the rung goes false while the timer is paused, the timer resets as normal.

1. If you change to Program mode, or the processor loses power before the instruction reaches the preset value, the following occurs:
 - timer enable (.EN) bit remains set
 - timer timing (.TT) bit remains set
 - accumulated (.ACC) value remains the same
2. Then when you switch back to Run mode or Test mode or power is restored, the following happens:

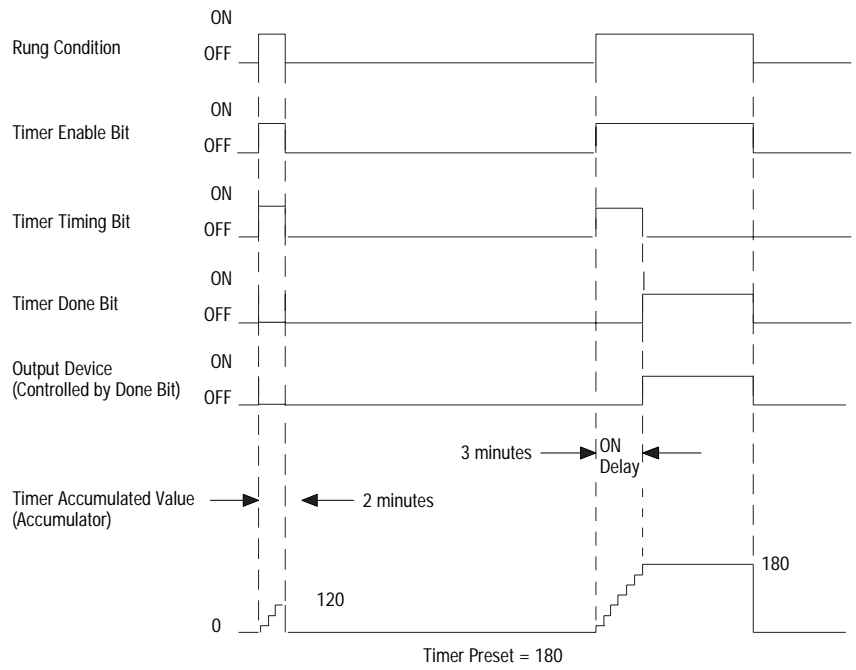
Condition:	Result:
If the rung is true:	.EN bit remains set .TT bit remains set .DN bit remains reset .ACC value is reset and starts counting up
If the rung is false:	.EN bit is reset .TT bit is reset .DN bit is reset .ACC value is reset

Figure 2.1
Example TON Ladder Diagram



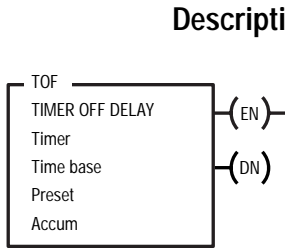
When bit I:012/10 is set, the processor starts T4:0. The accumulated value increments in 1-second intervals. T4:0.TT is set and output bit O:013/01 is set (the associated output device is energized) while the timer is timing. When the timer is finished (.ACC = .PRE) T4:0.TT is reset (so O:013/01 and the associated output device is de-energized) and T4:0.DN is set (so O:013/02 is set and the associated output device is energized). When the accumulated value reaches 180, the .DN bit is set. Or if the rung goes false, the timer is reset.

Figure 2.2
Example TON Timing Diagram



16649

Timer Off Delay (TOF)



Description: Use the TOF instruction to turn an output on or off after its rung has been off for a preset time interval. The TOF instruction starts accumulating time when the rung goes false and continues timing until one of the following conditions occur:

- the accumulated value equals its preset value
- the rung goes true
- a reset instruction resets the timer
- the SFC step goes inactive

The processor resets the accumulated value when the rung conditions go true, regardless of whether the timer timed out or not.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address the status bits by mnemonic.

This Bit:	Is Set When:	And Remains Set Until One of the Following Occurs:
Timer Enable .EN (bit 15)	the rung goes true	<ul style="list-style-type: none"> • the rung goes false • a reset instruction resets the timer • the SFC step goes inactive
Timer Timing Bit .TT (bit 14)	the rung goes false and the accumulated value is less than the preset	<ul style="list-style-type: none"> • the rung goes true • the .DN bit is set (.ACC = .PRE) • a reset instruction resets the timer • the associated SFC step goes inactive
Timer Done Bit .DN (bit 13)	the rung goes true	<ul style="list-style-type: none"> • the accumulated value is equal to the preset value

If you set the done bit .DN using an OTE instruction, for example, you can pause the timer. The .EN and .TT bits remain set, but the accumulated value does not increment. Timing resumes when you clear the .DN bit. If the rung goes false while the timer is paused, the timer resets as normal.

1. If you change to Program mode, or the processor loses power, or the processor fault interrupts the TOF instruction before it reaches the preset value, the following occurs:
 - timer enable (.EN) bit remains reset
 - timer timing (.TT) bit remains set
 - timer done (.DN) bit remains set
 - accumulated (.ACC) value remains the same
2. Then if you switch to Run mode or Test mode, the following happens:

Condition:	Result:
If the rung is true:	.EN bit is set .TT bit is reset .DN bit remains set .ACC value is cleared
If the rung is false:	.EN bit is reset .TT bit is reset .DN bit is reset .ACC value equals PRE value (the timer does not start timing)

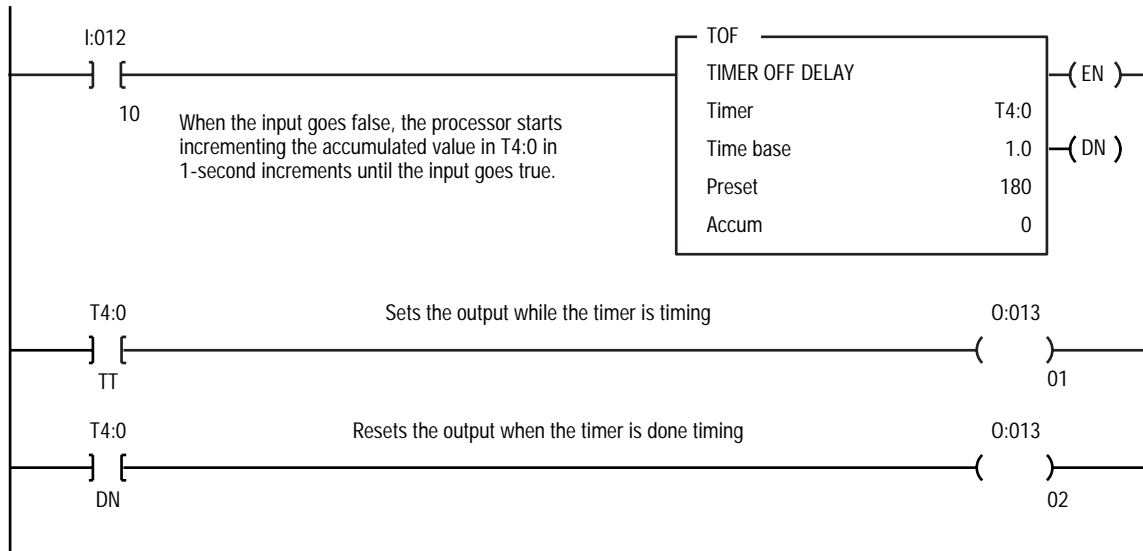


ATTENTION: Because the RES instruction resets the accumulated value, done bit and timing bits of a timing instruction, do not use the RES instruction to reset a TOF timer.

During prescan, the following happens:

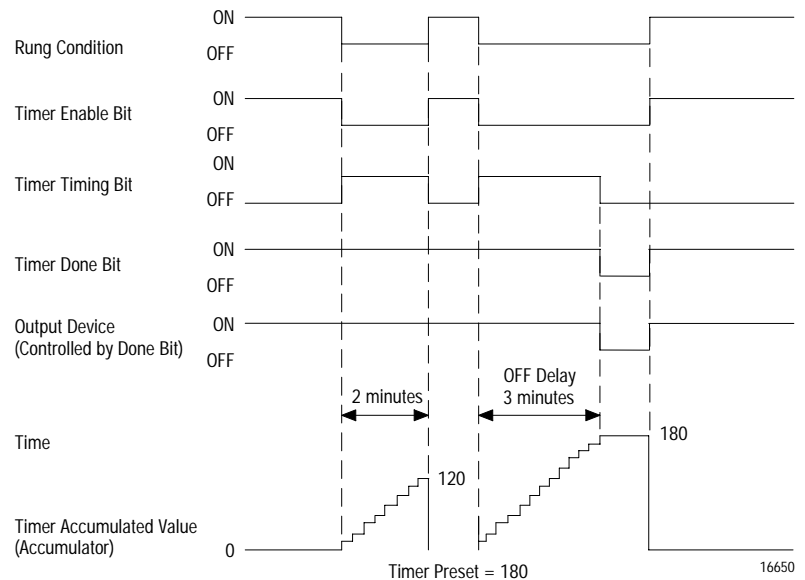
- timer timing (.TT) bit is cleared
- accumulated (.ACC) value is equal to the preset value

Figure 2.3
Example TOF Ladder Diagram

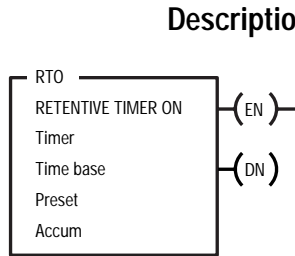


When bit I:012/10 is reset, the processor starts timer T4:0. The accumulated value increments by 1-second intervals as long as the rung remains false. T4:0.TT is set and output bit O:013/01 is set (the associated output device is energized) while the timer is timing. When the timer is finished (.ACC = .PRE), T4:0.TT is reset (so O:013/01 is reset and the associated output device is de-energized) and T4:0.DN is reset (so O:013/02 is reset and the associated output device is de-energized). When the accumulated value reaches 180 or when the rung conditions go true, the timer stops.

Figure 2.4
Example TOF Timing Diagram



Retentive Timer On (RTO)



Description:

Use the RTO instruction to turn an output on or off after its timer has been on for a preset time interval. The RTO instruction lets the timer stop and start without resetting the accumulated value.

The RTO instruction begins timing when its rung goes true. As long as the rung remains true, the timer updates the accumulated value each program scan, until it reaches the preset value. The RTO instruction retains its accumulated value even if one of the following occurs:

- the rung goes false
- you change to Program mode
- the processor faults or loses power
- the SFC step goes inactive

When the processor resumes operation or the rung goes true, timing continues from the retained accumulated value. By retaining its accumulated value, retentive timers measure the cumulative period during which its rung is true.

Important: To reset the retentive timer’s accumulated value and status bits after the RTO rung goes false, you must program a reset instruction RES with the same address in another rung.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address the status bits by mnemonic.

This Bit:	Is Set When:	Indicates:	And Remains Set Until One of the Following Occurs:
Timer Enable Bit .EN (bit 15)	the rung goes true	that a timing operation is in progress	<ul style="list-style-type: none"> • the rung goes false • a reset instruction resets the timer
Timer Timing Bit .TT (bit 14)	the rung goes true	that a timing operation is in progress	<ul style="list-style-type: none"> • the rung goes false • the .DN bit is set • the accumulated value is equal to the preset value (.ACC=.PRE) • a reset instruction resets the timer
Timer Done Bit .DN (bit 13)	the accumulated value is equal to the preset value	that a timing operation is complete	<ul style="list-style-type: none"> • the .DN bit is reset with the RES instruction.

If you set the done bit .DN using an OTE instruction, for example, you can pause the timer. The .EN and .TT bits remain set, but the accumulated value does not increment. Timing resumes when you clear the .DN bit. If the rung goes false while the timer is paused, the timer resets as normal.

1. If you change to Program mode, or the processor loses power, or a processor fault interrupts the RTO instruction, the following occurs:
 - timer enable (.EN) bit remains set
 - timer timing (.TT) bit remains set
 - accumulated (.ACC) value remains the same
2. When you switch back to Run mode or Test mode, the following happens:

Condition:	Result:
If the rung is true:	.EN bit remains set .TT bit remains set .ACC value continues timing
If the rung is false:	.EN bit is reset .TT bit is reset .DN bit remains the same .ACC value remains the same

Figure 2.5
Example RTO Ladder Diagram

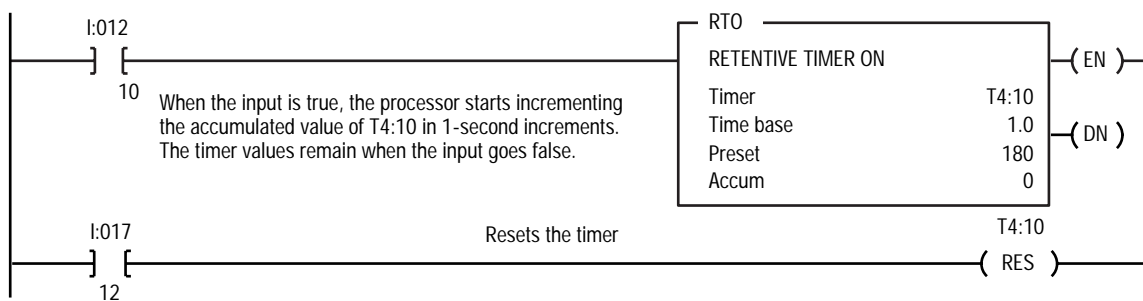
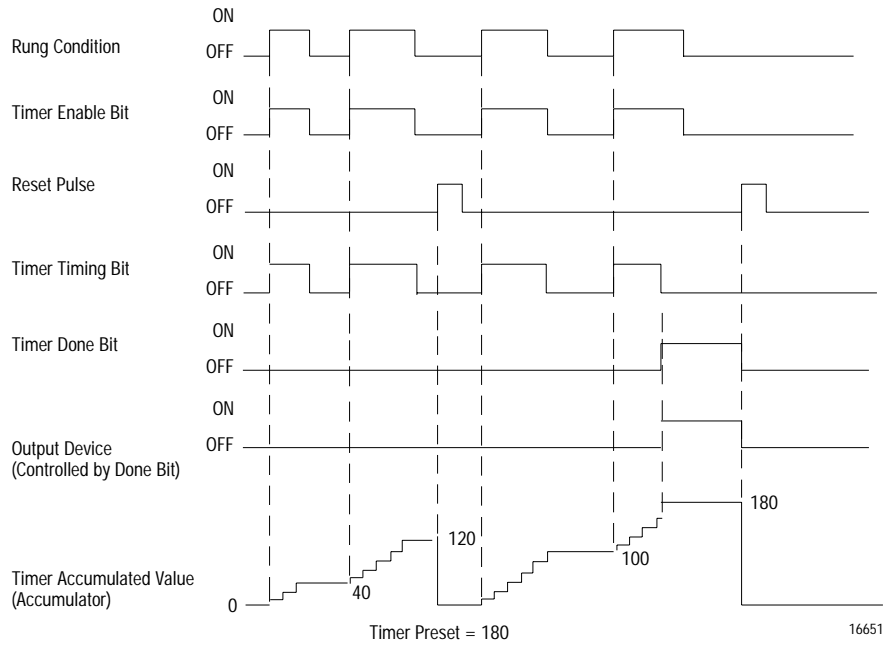
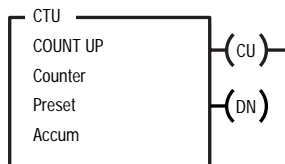


Figure 2.6
Retentive Timer Timing Diagram



Using Counters

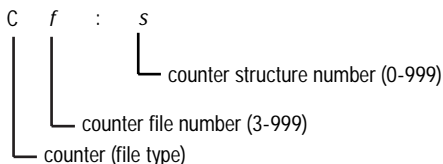


Before using counter instructions, you need to understand the parameters that you enter.

Entering Parameters

To program a counter instruction, provide the processor with the following information:

- **Counter** is the counter control address in the counter (C) area of data storage. Use the following address format:



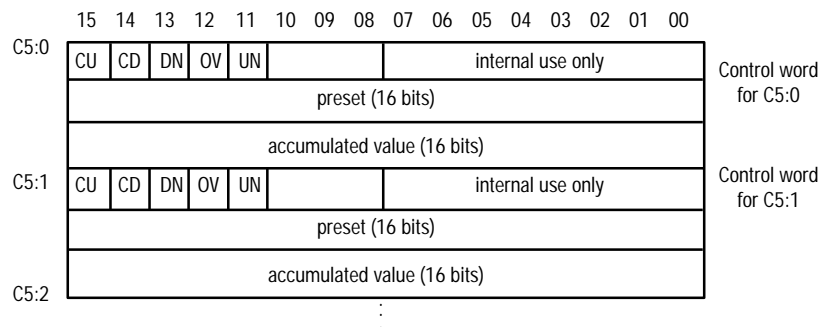
Important: You can use any counter file number from 3 to 999; however, the default counter file number is 5. If you want to specify a counter file number as any file between 3 and 8 (other than the default 5), you must first delete the entire default file for that number, and then create the counter file. For example, if you want a counter file number as file 3, you must first delete the entire default binary file and then create the counter file as file 3.

To access a counter status bit, preset value, or accumulated value, use the following address format:

Status Bit	Preset	Accumulated Value
<i>Cf.s.bb</i>	<i>Cf.s.PRE</i>	<i>Cf.s.ACC</i>

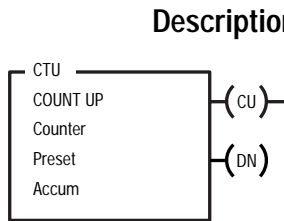
The *bb* is a status bit mnemonic, such as .DN

Important: The processor stores counter status bits and the preset and accumulated values in a storage structure (48 bits – three 16-bit words) in a counter file (C) in the data table.



- **Preset** specifies the value which the counter must reach before it sets the done bit .DN. Enter a preset value from –32,768 up to +32,767. The preset value is stored as a 16-bit integer value. Negative values are stored in twos complement form.
- **Accumulated Value** is the current count based on the number of times the rung goes from false to true. The accumulated value is stored as a 16-bit integer value. Negative values are stored in twos complement form. The range of the accumulated value is –32,768 to +32,767. Typically, you enter a zero value when programming counter instructions. If you enter a non-zero value, the instruction starts counting from that value. If the counter is reset, the accumulated value is set to zero.

Count Up (CTU)



Description:

The CTU instruction counts upward over a range of $-32,768$ to $+32,767$. Each time the rung goes from false to true, the CTU instruction increments the accumulated value by one count. When the accumulated value equals or exceeds the preset value, the CTU instruction sets a done bit .DN, which your ladder program can use to initiate some action, such as controlling a storage bit or an output device.

The accumulated value of a counter is retentive. The count is retained until reset by a reset instruction (RES) that has the same address as the counter.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs the CTU instruction. You address the status bits by mnemonic.

This Bit:	Is Set:	And Remains Set Until One of the Following Occurs:
Count Up Enable Bit .CU (bit 15)	when the rung goes true to indicate the instruction has increased its count Note: During prescan, this bit is set to prevent a false count when the program scan begins.	<ul style="list-style-type: none"> the rung goes false a RES instruction resets the .DN bit
Count Up Done Bit .DN (bit 13)	when the accumulated value is greater than or equal to the preset value	<ul style="list-style-type: none"> the accumulated value counts below the preset, either by using a CTD instruction to count down or changing the accumulated value a RES instruction resets the .DN bit
Count Up Overflow Bit .OV (bit 12)	when the up counter has exceeded the upper limit of $+32,767$ and has wrapped around to $-32,768$. The CTU counts up from there.	<ul style="list-style-type: none"> a RES instruction resets the .DN bit counting back down to $32,767$ with a CTD instruction with the same address



ATTENTION: Place critical counters outside an MCR zone or jumped sections of ladder program to guard against invalid results that could lead to damaged equipment or personnel injury.

Figure 2.7
Example CTU Ladder Diagram

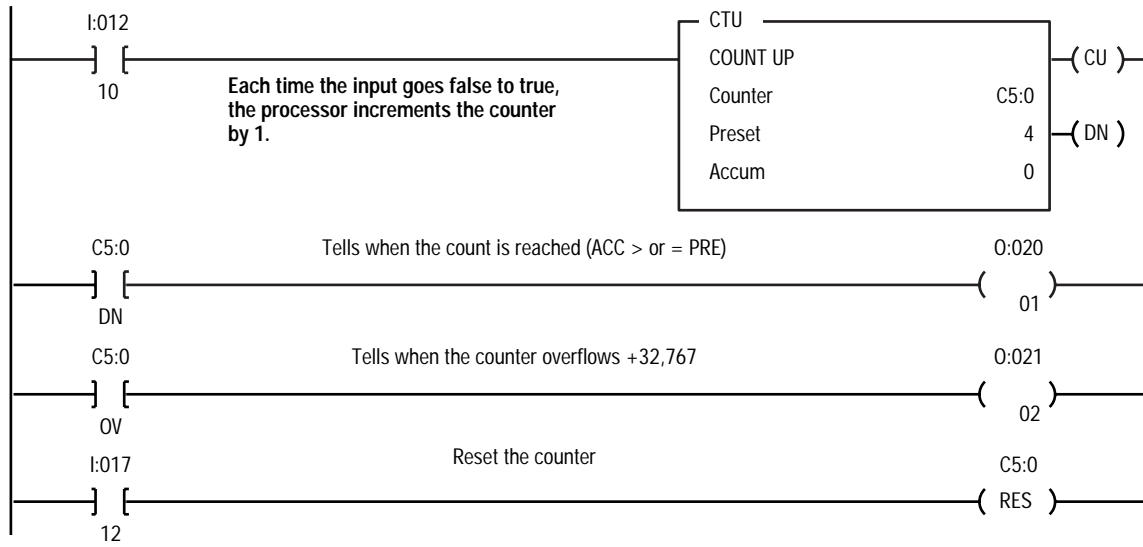
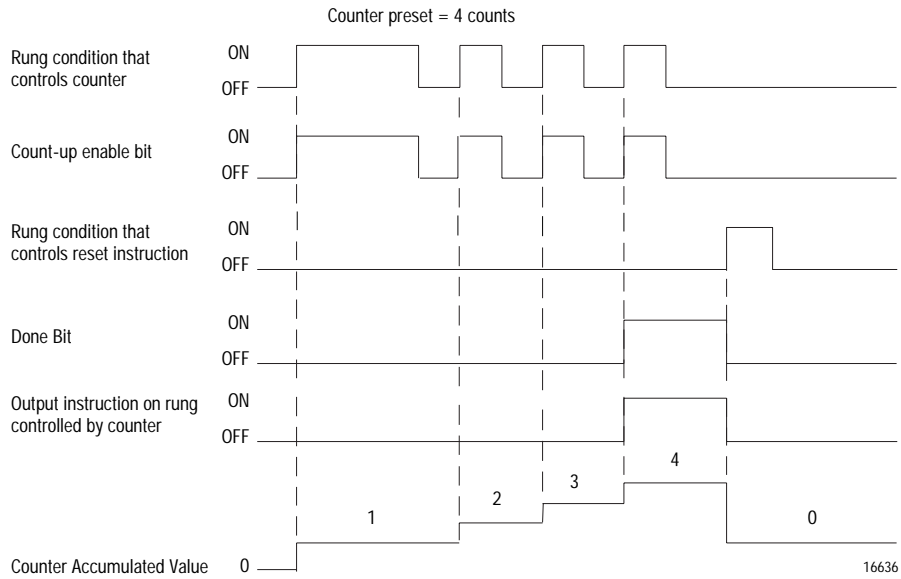
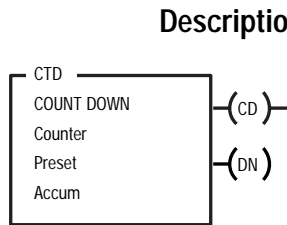


Figure 2.8
Example CTU Timing Diagram



Count Down (CTD)



Description:

The CTD instruction counts downward over a range of +32,767 to -32,768. Each time the rung goes from false to true, the CTD instruction decrements the accumulated value by one count. The done bit .DN is set as long as the accumulated value is greater than or equal to the preset value. When the accumulated value is less than the preset value, the done bit .DN is reset, which your ladder program can use to initiate some action, such as controlling a storage bit or an output device.

The accumulated value of a counter is retentive. The count is retained until reset by a reset instruction (RES) that has the same address as the CTD instruction.

Using Status Bits

Examine status bits in the ladder program to trigger some event. The processor changes the states of status bits when the processor runs this instruction. You address the status bits by mnemonic.

This Bit:	Is Set:	And Remains Set Until One of the Following Occurs:
Count Down Enable Bit .CD (bit 14)	when the rung goes true to indicate that the counter is enabled as a down-counter. Note: During prescan, this bit is set to prevent a false count when the program scan begins.	<ul style="list-style-type: none"> the rung goes false a RES instruction resets the .DN bit
Count Down Done Bit .DN (bit 13)	when the accumulated value is greater than or equal to the preset value.	<ul style="list-style-type: none"> the accumulated value counts below the preset another instruction changes the accumulated value a RES instruction resets the .DN bit
Count Down Underflow Bit (.UN) (Bit 11)	by the processor to show that the down counter went below the lower limit of -32,768 and has wrapped around to +32,767. The CTD instruction counts down from there.	<ul style="list-style-type: none"> a RES instruction resets the .DN bit count back up to -32,768 with a CTU instruction



ATTENTION: Place critical counters outside an MCR zone or jumped sections of ladder program to guard against invalid results that could lead to damaged equipment or personnel injury.

Figure 2.9
Example CTD Ladder Diagram

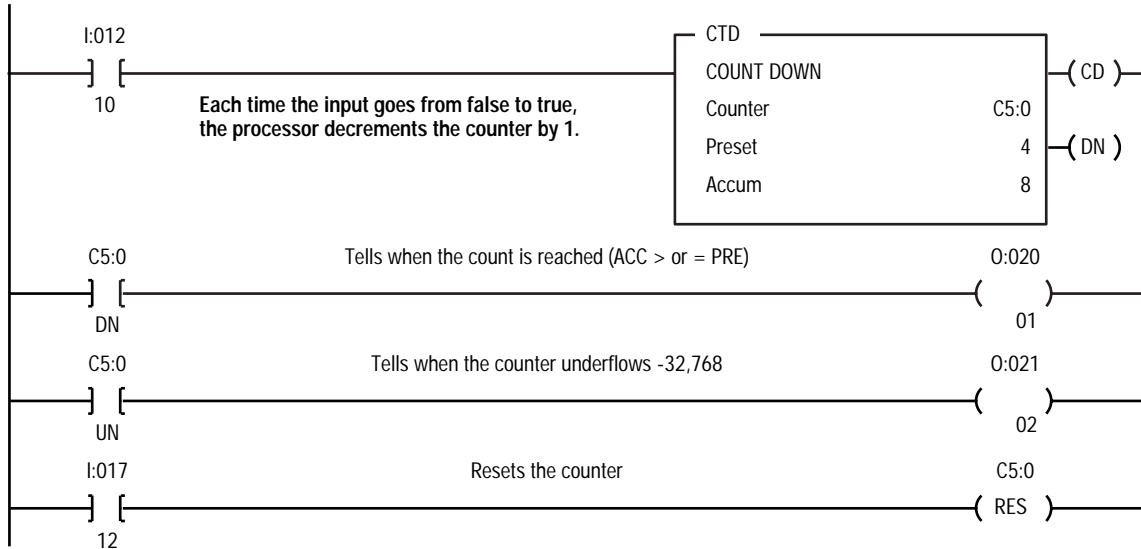


Figure 2.10
Example CTD Timing Diagram

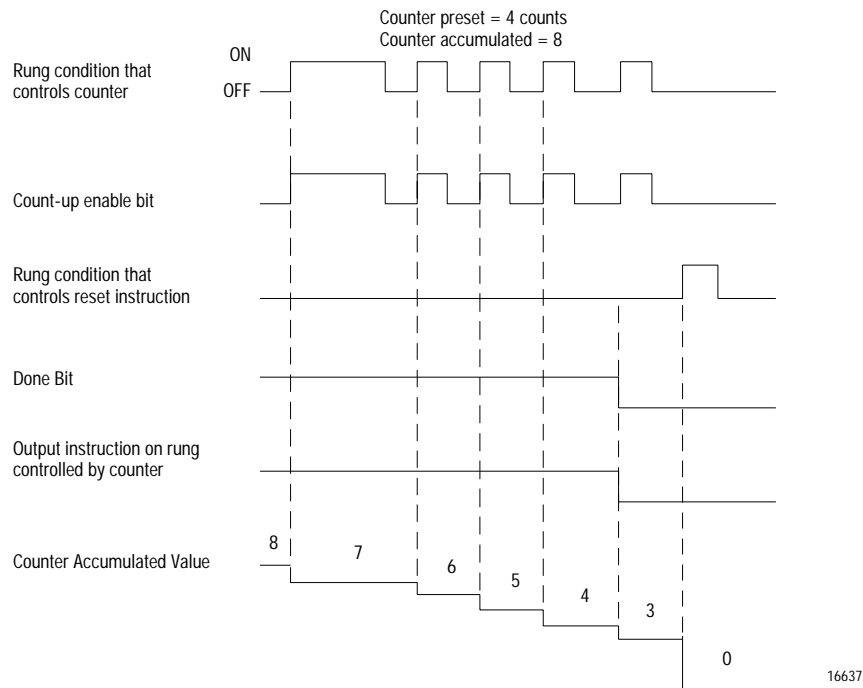


Figure 2.11
Example CTU and CTD Logic Diagram

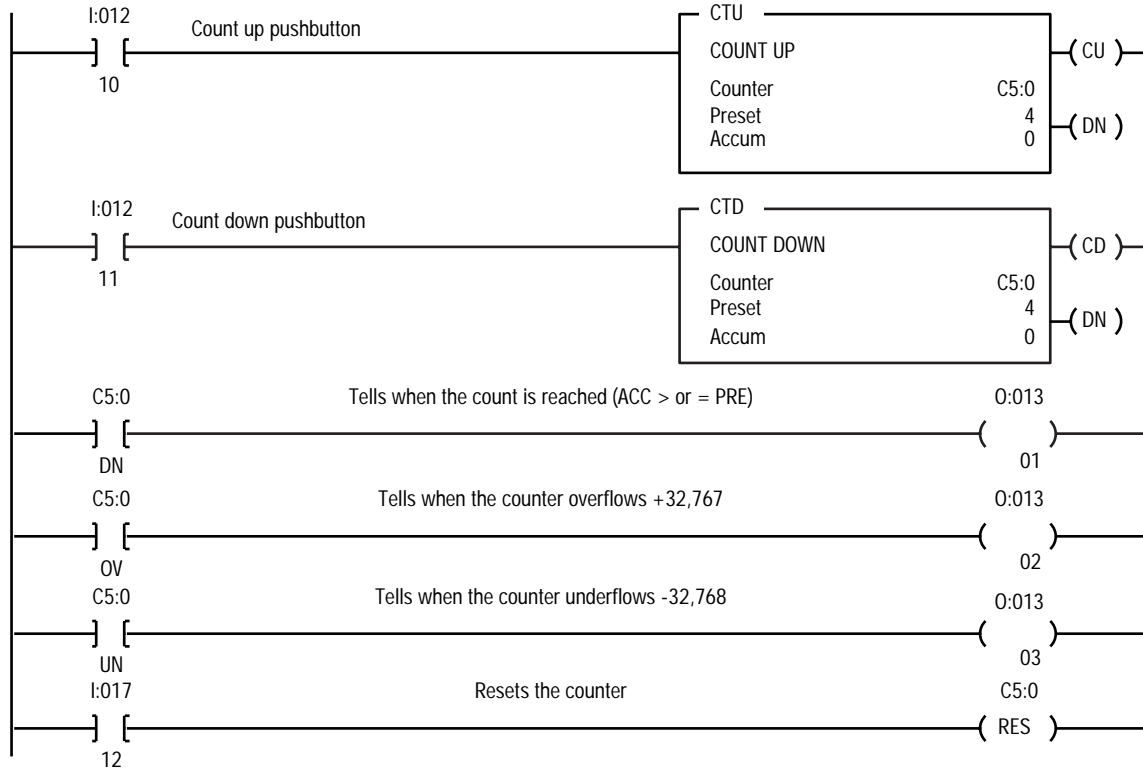
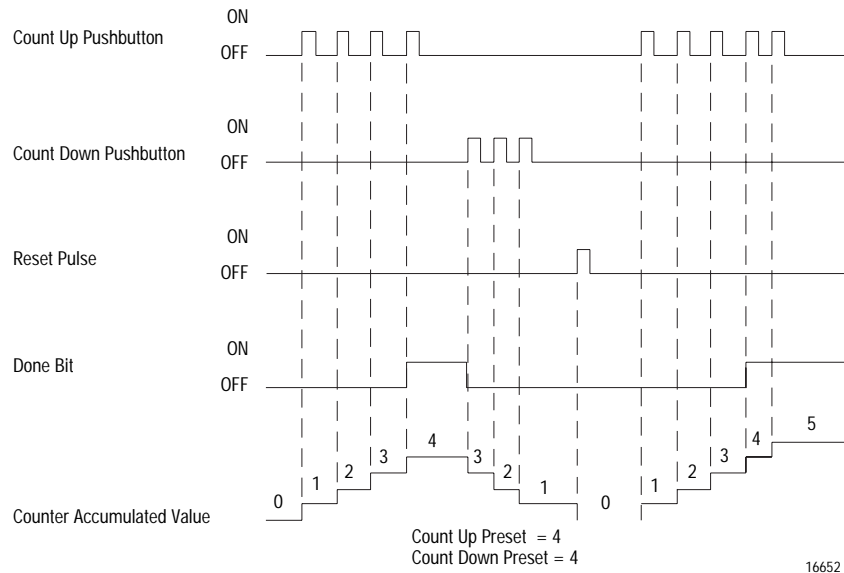


Figure 2.12
Example CTU and CTD Timing Diagram



Timer and Counter Reset (RES)

Description: The RES instruction is an output instruction that resets a timer or counter. The RES instruction executes when its rung is true.

—(RES)—

When Using a RES Instruction for a:	The Processor Resets the:
Timer (Do not use a RES instruction for a TOF.)	.ACC value .EN bit .TT bit .DN bit
Counter	.ACC value .EN bit .OV or .UN bit .DN bit

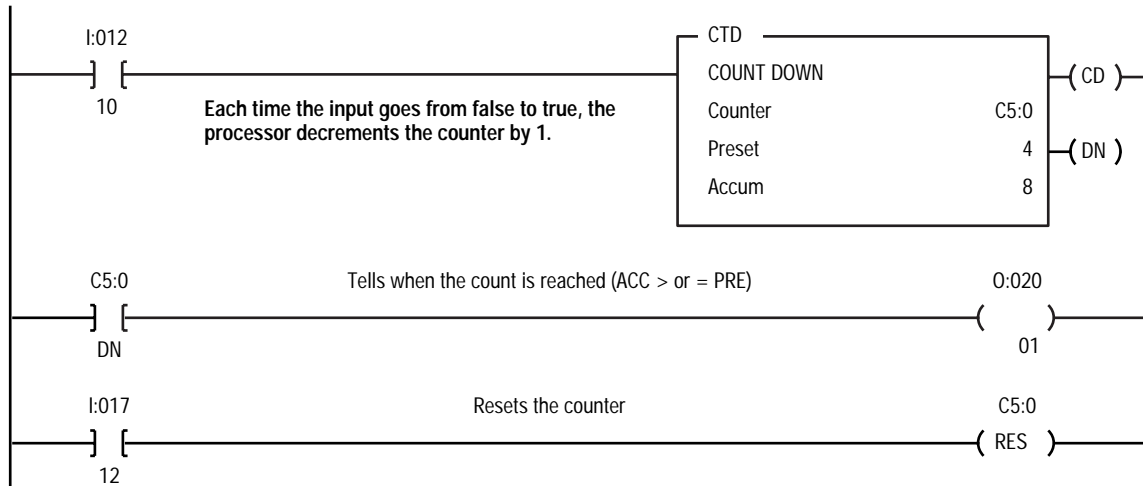
If the counter rung is enabled, the CU or CD bit will be reset as long as the RES instruction is enabled.

Important: You can use a negative preset value in a CTU or CTD instruction if you intend to use the RES instruction. However, note that the RES instruction sets the accumulated value to zero, which may set the .DN bit and prevent the CTU or CTD instruction from operating the next time it is enabled.



ATTENTION: Because the RES instruction resets the accumulated value, .DN bit and .TT bit of a timing instruction, do not use the RES instruction to reset a TOF instruction; unpredictable machine operation or injury to personnel may occur.

Figure 2.13
Example RES Ladder Diagram



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

Using Compare Instructions

The comparison instructions let you compare values using an expression or a specific comparison instruction. Table 3.A lists the available compare instructions.

Table 3.A
Available Compare Instructions

If You Want to:	Use the Instruction:	On Page:
Compare values based on an expression	CMP	3-2
Test whether two values are equal	EQU	3-5
Test whether one value is greater than or equal to a second value	GEQ	3-5
Test whether one value is greater than a second value	GRT	3-6
Test whether one value is less than or equal to a second value	LEQ	3-6
Test whether one value is less than a second value	LES	3-7
Test whether one value is between two other values	LIM	3-7
Pass two values through a mask and test whether they are equal	MEQ	3-9
Test whether one value is not equal to a second value	NEQ	3-10

Important: You can compare values of different data types, such as floating point and integer. You should use BCD and ASCII values for display purposes. If you enter BCD or ASCII values, the processor treats those values as integers. For example, if the value at N7:2 is 10 (decimal) and the value at D9:3 is 10 (BCD), the comparison of N7:2 = D9:3 evaluates as false. The 10 in BCD translates to 0000 0000 0001 0000; the 10 in decimal translates to 0000 0000 0000 1010.

The parameters you enter are program constants or logical addresses of the values you want to compare.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Arithmetic Status Flags

The arithmetic status flags are in word 0 bits 0-3 in the processor status file (S). Monitor these bits if you perform an arithmetic function within the CMP instruction. Table 3.B lists the status bits:

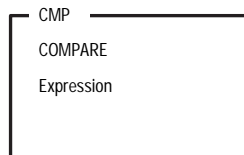
Table 3.B
Arithmetic Status Bits

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

Compare (CMP)

The CMP instruction compares values and performs logical comparisons.

Description:



The CMP instruction is an input instruction that performs a comparison on arithmetic operations you specify in the expression. When the processor finds the expression is true, the rung goes true. Otherwise, the rung is false. With Enhanced PLC-5 processors, you can enter multiple operands (complex expression).

The execution time of a CMP instruction is longer than the execution time of one of the other comparison instructions (e.g., GRT, LEQ, etc.). A CMP instruction also uses more words in your program file than the corresponding comparison instruction.

Entering the CMP Expression

The expression defines the operations you want to perform. Define the expression with operators and addresses or program constants. With Enhanced PLC-5 processors, you can enter complex expressions. Table 3.C lists valid operations for an expression; the following list provides guidelines for writing expressions.

- Operators (symbols) define the operations
- Addresses can be direct, indirect, or indexed address(es) (must be word level)
- With Enhanced PLC-5 processors, program constants can be integer or floating-point numbers (if you enter octal values, use a leading &O; if you enter hexadecimal values, use a leading &H; if you enter binary values, use a leading &B)

Table 3.C
Valid Operations for Use in a CMP Expression

Type	Operator	Description	Example Operation
Comparison	=	equal to	if A = B, then ...
	<>	not equal to	if A <> B, then ...
	<	less than	if A < B, then ...
	<=	less than or equal to	if A <= B, then ...
	>	greater than	if A > B, then ...
	>=	greater than or equal to	if A >= B, then ...
Arithmetic	+	add	2 + 3 Enhanced PLC-5 processor: 2 + 3 + 7
	-	subtract	12 - 5
	*	multiply	5 * 2 PLC-5/30, -5/40, -5/60, -5/80: 6 * (5 * 2)
	(vertical bar)	divide	24 6
	-	negate	- N7:0
	SQR	square root	SQR N7:0
	**	exponential (x to the power of y)	10**3 (Enhanced PLC-5 processors only)
Conversion	FRD	convert from BCD to binary	FRD N7:0
	TOD	convert from binary to BCD	TOD N7:0

Determining the Length of an Expression

Enhanced PLC-5 processors support complex instructions (up to a total of 80 characters, including spaces and parentheses). Depending on the operator, the processor inserts characters before/after the operator in your expression to format the expression for easier interpretation. Use Table 3.D to determine the number of characters each operator uses in an expression.

Important: You cannot enter floating point numbers in scientific notation with negative exponents in complex expressions. Instead, use the decimal equivalent or put the number in a floating point file and use the data address in the complex expression.

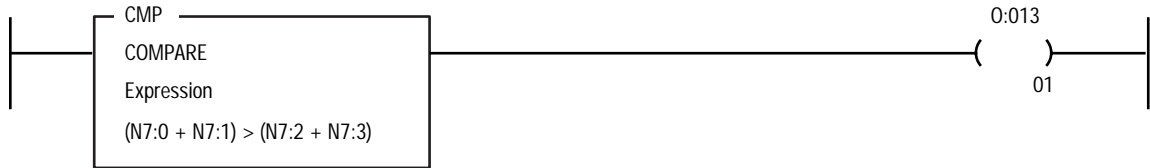
With the CMP instruction, a maximum of 80 characters of the expression can be displayed. If the expression you enter is near this 80 character maximum, when you accept the rung containing the instruction, the processor may expand it beyond 80 characters. When you try to edit the expression, only the first 80 characters are displayed and the rung is displayed as an error rung. The processor does contain the complete expression, however, and the instruction runs properly.

To avoid this display problem, export the processor memory file and make your edits in the PC5 text file. Then import this text file. For more information on importing/exporting processor memory files, see your programming manual.

Table 3.D
Character Lengths for Operators

This Operation:	Using this Operator:	Uses this Number of Characters:
math binary	+, -, *,	3
	OR, **	4
	AND, XOR	5
math unary	- (negate)	2
	LN	3
	FRD, TOD, DEG, RAD, SQR, NOT, LOG, SIN, COS, TAN, ASN, ACS, ATN	4
comparative	=, <, >	3
	<>, <=, >=	4

Example:

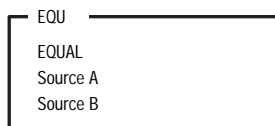


The CMP instruction tells an Enhanced PLC-5 processor: if the sum of the values in N7:0 and N7:1 is greater than the sum of the values in N7:2 and N7:3, set output bit O:013/01. (The total number of characters used in this expressions is 3.)

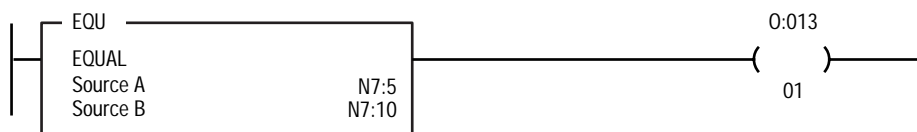
For more information on entering complex expressions, see chapter 4.

Equal to (EQU)

Description: Use the EQU instruction to test whether two values are equal. Source A and Source B can either be values or addresses that contain values.



Example:

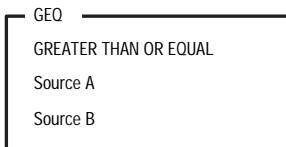


If the value in N7:5 is equal to the value in N7:10, set output bit 0:013/01.

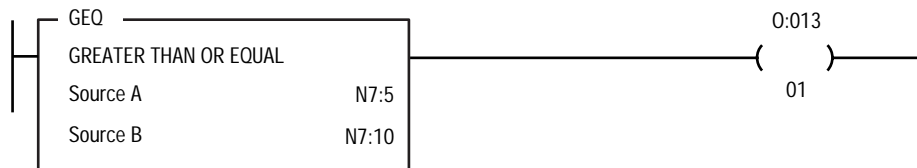
Floating point values are rarely absolutely equal. If you need to determine the equality of floating point values, use the LIM instruction (instead of the EQU). For information on the LIM instruction, see page 3-7.

Greater than or Equal to (GEQ)

Description: Use the GEQ instruction to test whether one value (Source A) is greater than or equal to another value (Source B). Source A and Source B can be values or addresses that contain values.



Example:

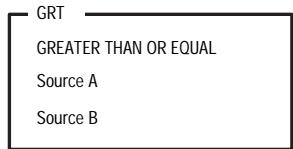


If the value in N7:5 is greater than or equal to the value in N7:10, set output bit 0:013/01.

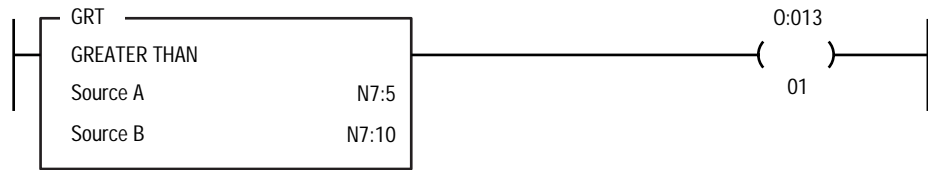
Greater than (GRT)

Description:

Use the GRT instruction to test whether one value (Source A) is greater than another value (Source B). Source A and Source B can either be values or addresses that contain values.



Example:

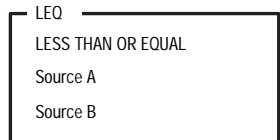


If the value in N7:5 is greater than the value in N7:10, set output bit 0:013/01.

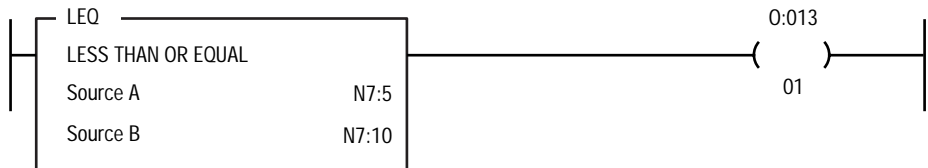
Less than or Equal to (LEQ)

Description:

Use the LEQ instruction to test whether one value (Source A) is less than or equal to another value (Source B). Source A and Source B can either be values or addresses that contain values.



Example:

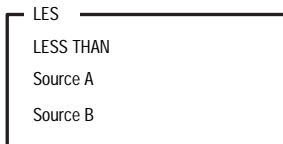


If the value in N7:5 is less than or equal to the value in N7:10, set output bit 0:013/01.

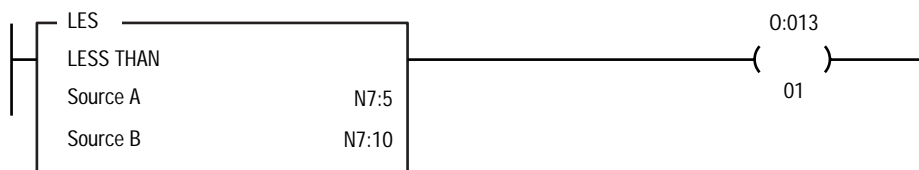
Less than (LES)

Description:

Use the LES instruction to test whether one value (Source A) is less than another value (Source B). Source A and Source B can be values or addresses that contain values.



Example:

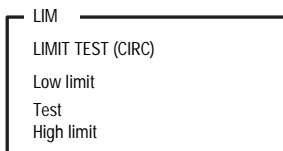


If the value in N7:5 is less than the value in N7:10, set output bit 0:013/01.

Limit Test (LIM)

Description:

The LIM instruction is an input instruction that tests for values inside of or outside of a specified range. The instruction is false until it detects that the test value is within certain limits. Then the instruction goes true. When the instruction detects that the test value goes outside certain limits, it goes false.



You can use the LIM instruction to test if an analog input value is within specified limits.

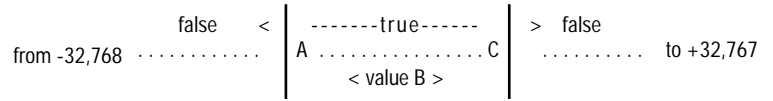
Entering Parameters

To program the LIM instruction, you must provide the processor with the following:

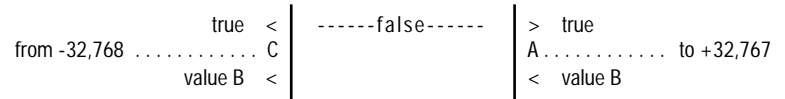
Parameter:	Definition:
Low Limit	a constant or an address from which the instruction reads the lower range of the specified limit range. The address contains an integer or floating-point value.
Test Value	the address that contains the integer or floating-point value you examine to see whether the value is inside or outside the specified limit range.
High Limit	a constant or an address from which the instruction reads the upper range of the specified limit range. The address contains an integer or floating-point value.

LIM Example Using Integer:

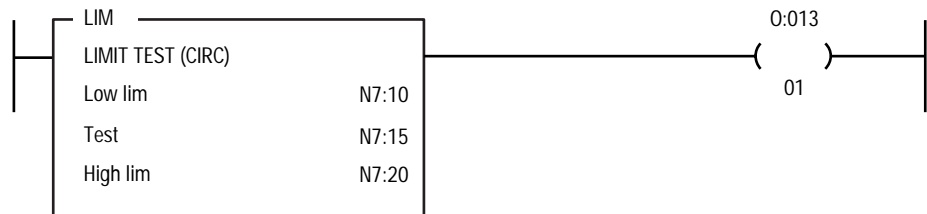
- **If value Low Limit ≤ value High Limit:** When the processor detects that the value of B (Test) is equal to or between limits, the instruction is true; if value Test is outside the limits, the instruction is false.



- **If value Low Limit ≥ value High Limit:** When the processor detects that the value of Test is equal to or outside the limits, the instruction is true; if value Test is between, but not equal to either limit, the instruction is false.



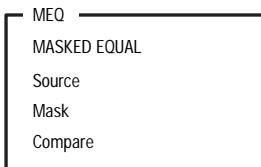
Example (when the Low Limit is less than the High Limit):



If the value in N7:15 is greater than or equal to the value in N7:10 and less than or equal to the value in N7:20, set output bit O:013/01.

Mask Compare Equal to (MEQ)

Description:



The MEQ instruction is an input instruction that compares a value from a source address with data at a compare address, and allows portions of the data to be masked. If the data at the source address matches the data at the compare address bit-by-bit (less masked bits), the instruction is true. The instruction goes false as soon as it detects a mismatch.

You can use the MEQ instruction to extract (for comparison) bit data such as status or control bits from an element that contains bit and word data.

Entering Parameters

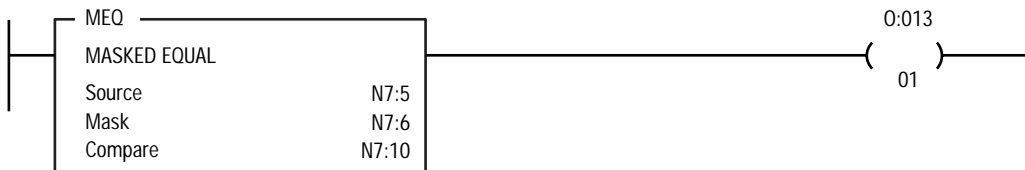
To program the MEQ instruction, you must provide the processor with the following:

Parameter:	Definition:
Source	a program constant or data address from which the instruction reads an image of the value. The source remains unchanged.
Mask	specifies which bits to pass or block. A mask passes data when the mask bits are set (1); a mask blocks data when the mask bits are reset (0). The mask must be the same element size (16-bits) as the source and compare address. In order for bits to be compared, you must set (1) mask bits; bits in the compare address that correspond to zeros (0) in the mask are not compared. If you want the ladder program to change the mask value, store the mask at a data address. Otherwise, enter a hexadecimal value for a constant mask value. If you enter a hexadecimal value that starts with a letter (such a F800), enter the value with a leading zero. For example, type 0F800
Compare	specifies whether you want the ladder program to vary the compare value, or a program constant for a fixed reference. Use 16-bit elements, the same as the source.

Example:

```

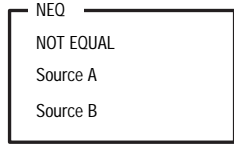
Source      01010101 01011111
Mask        11111111 11110000
Compare     01010101 0101xxxx
Result      The instruction is true because
              reference bits xxxx are not compared.
    
```



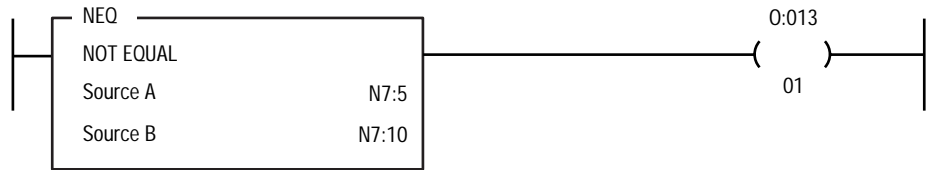
The processor passes the value in N7:5 through the mask in N7:6. It then passes the value in N7:10 through the mask in N7:6. If the two masked values are equal, set output bit 0:013/01

Not Equal to (NEQ)

Description: Use the NEQ instruction to test whether two values are not equal. Source A and Source B can be values or addresses.



Example:



If the value in N7:5 is not equal to the value in N7:10, set output bit O:013/01.

Compute Instructions CPT, ACS, ADD, ASN, ATN, AVE, CLR, COS, DIV, LN, LOG, MUL, NEG, SIN, SRT, SQR, STD, SUB, TAN, XPY

Using Compute Instructions

The compute instructions evaluate arithmetic operations using an expression or a specific arithmetic instruction. Table 4.A lists the available compute instructions.

Table 4.A
Available Compute Instructions

If You Want to:	Use this Instruction:	Found on Page:
Evaluate an expression	CPT	4-5
Take the arc cosine of a number	ACS*	4-11
Add two values	ADD	4-12
Take the arc sine of a number	ASN*	4-13
Take the arc tangent of a number	ATN*	4-14
Calculate the average for a set of values	AVE*	4-15
Clear an address word (set all bits to zero)	CLR	4-17
Take the cosine of a number	COS*	4-18
Divide two values	DIV	4-19
Take the natural log of a number	LN*	4-20
Take the log of a number	LOG*	4-21

* Only Enhanced PLC-5 processors support this instruction.

(Continued)

If You Want to:	Use this Instruction:	Found on Page:
Multiply two values	MUL	4-22
Take the opposite sign of a value	NEG	4-23
Take the sine of a number	SIN*	4-24
Take the square root of a value	SQR	4-25
Sort a set of values into ascending order	SRT*	4-26
Calculate the standard deviation for a set of values	STD*	4-28
Subtract two values	SUB	4-31
Take the tangent of a number	TAN*	4-32
Raise a number to a power	XPY*	4-33

* Only Enhanced PLC-5 processors support this instruction.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Arithmetic Status Flags

The arithmetic status flags are in word 0 bits 0-3 in the processor status file (S). Table 4.B lists the status bits:

Table 4.B
Arithmetic Status Bits

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

Data Types and the Compute Instruction

You can compute values of different data types, such as floating point and integer. If you use a floating-point as the source, use a floating-point as the destination; otherwise, the destination value will be rounded.

You should use BCD and ASCII values for display purposes. If you enter BCD or ASCII values, the processor treats those values as integers.

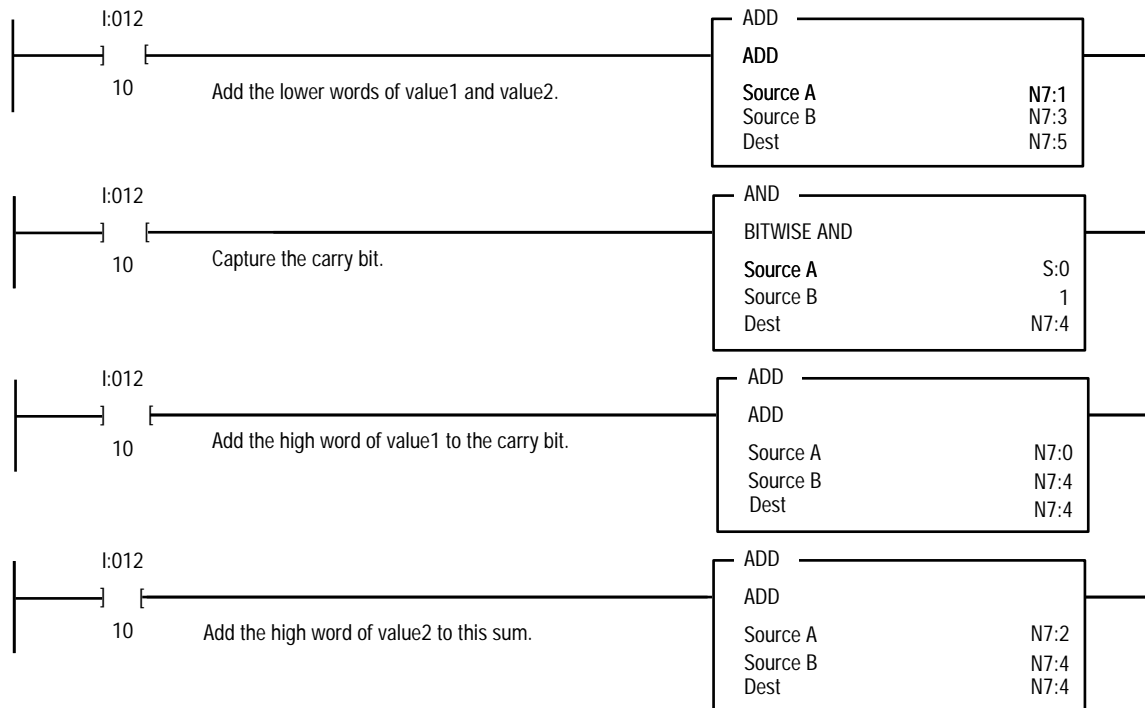
The parameters you enter are program constants or logical addresses of the values you want.

If You are Using this Processor:	The Processor Rounds:
Classic PLC-5	the final value of a mathematical operation before storing the final result. The processor rounds to the nearest whole number: The processor rounds values of 0.5-0.9 up to the next whole number; the processor rounds values of 0.1- 0.4 down to the closest whole number. If this value is greater than 32,767 or less than -32,768, the overflow status bit is set.
Enhanced PLC-5	down if the value is < 0.5, up if the value is > 0.5, and to the nearest even number if the value is = 0.5. If this value is greater than 32,767 or less than -32,768, the processor "wraps" negative (32,767, -32,768, -32,767, -32,766, etc.). For example, if you have an ADD instruction with a result greater than 32,767, the overflow bit is set, the sign bit is set, and the result is negative: $32,767 + 5 = -32,764$.

Important: If you are using an Enhanced PLC-5 processor and an arithmetic operation generates an overflow, the upper bits are lost, but the lower bits are correct. If you then perform a logical operation on the lower word (AND or OR), you can get the proper result. Also, using the carry bit, you can do multi-word arithmetic (i.e., add two 32-bit words).

For example, if: value1 = N7:0 and N7:1
 value2 = N7:2 and N7:3
 result = N7:4 and N7:5

and you want to add value1 to value2, your ladder program would be:



Using Floating Point Data Types

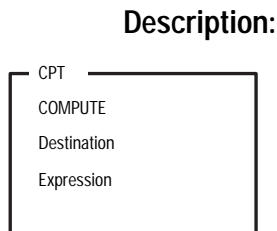
For an Enhanced PLC-5 processor, if you use floating point data types and the result is too large or if it is undefined (i.e. natural log of 0), the processor sets the overflow bit.

If the result is too large, a !+INF! is displayed; if the result is too small, a !-INF! is displayed. If the value is not a number, !NAN! is displayed.

Important: If you are using floating point and the number is greater than 32,767 or less than -32,768, you must use a decimal point. If you do not use a decimal point, the error INVALID OPERAND appears.

When you use complex expressions, if any operand is floating point, the entire expression is evaluated as floating point. See the example in the "Expression Examples" section on page 4-10 for more information.

Compute (CPT)



The CPT instruction performs copy, arithmetic, logical, and conversion operations.

The CPT instruction is an output instruction that performs the operations you define in the expression, and writes the result into the destination address. The CPT instruction can also copy data from one address to another and automatically converts the data type at the source address to the data type you specify in the destination address.

The execution time of a CPT instruction is longer than the execution time of an arithmetic, logic or move instruction (i.e., ADD, AND, MOV, etc.). The CPT instruction also uses more words in your program file.

After each CPT instruction is performed, the arithmetic status bits in the status file of the data table are updated the same as the corresponding arithmetic, logic or move instruction. For example, refer to the description of the ADD instruction to see how the status bits are updated after a CPT (add) instruction is executed.

Entering the CPT Expression

The expression defines the operations you want to perform. You define the expression with operators and addresses or program constants. With Enhanced PLC-5 processors, you can enter complex expressions. Table 4.C lists valid operations for an expression; the following list provides guidelines for writing expressions:

- Operators (symbols) define the operations
- Addresses can be direct or indirect logical address(es) (must be element or bit level)
- With Enhanced PLC-5 processors, program constants can be integer or floating-point numbers (if you enter octal values, use a leading &O; if you enter hexadecimal values, use a leading &H)
- Expressions can only total 80 characters, including spaces and parentheses

Table 4.C
Valid Operations for Use in a CPT Expression

Type	Operator	Description	Example Operation
Copy	none	copy from A to B	enter source address in the expression enter destination address in destination
Clear	none	set a value to zero	0 (enter 0 for the expression)
Arithmetic	+	add	2 + 3 2 + 3 + 7 (Enhanced PLC-5 processors)
	-	subtract	12 - 5 (12 - 5) - 7 (Enhanced PLC-5 processors)
	*	multiply	5 * 2 6 * (5 * 2) (Enhanced PLC-5 processors)
	(vertical bar)	divide	24 6 (24 6) * 2 (Enhanced PLC-5 processors)
	-	negate	- N7:0
	SQR	square root	SQR N7:0
	**	exponential * (x to the power of y)	10**3
	LN	natural log *	LN F8:20
	LOG	log to the base 10*	LOG F8:3
	Trigonometric	ACS	arc cosine*
ASN		arc sine*	ASN F8:20
ATN		arc tangent *	ATN F8:22
COS		cosine*	COS F8:14
SIN		sine*	SIN F8:12
TAN		tangent*	TAN F8:16
Bitwise	AND	bitwise AND	D9:3 AND D10:4
	OR	bitwise OR	D10:4 OR D10:5
	XOR	bitwise exclusive OR	D9:5 XOR D10:4
	NOT	bitwise complement	NOT D9:3
Conversion	FRD	convert from BCD to binary	FRD N7:0
	TOD	convert from binary to BCD	TOD N7:0
	DEG	convert radians to degrees*	DEG F8:8
	RAD	convert degrees to radians*	RAD F8:10

* Available in Enhanced PLC-5 processors only.

Determining the Length of an Expression

With Enhanced PLC-5 processors, you can enter complex instructions (up to a total of 80 characters, including spaces and parentheses). Depending on the operator, the processor inserts characters before/after the operator in your expression to format the expression for easier interpretation. Use Table 4.D below to determine the number of characters each operator uses in an expression.

With the CPT instruction, a maximum of 80 characters of the expression are displayable. If the expression you enter is near this 80 character maximum, when you accept the rung containing the instruction, the processor may expand it beyond 80 characters. When you try to edit the expression, only the first 80 characters are displayed and the rung is displayed as an error rung. The processor does contain the complete expression, however, and the instruction runs properly.

To work around this display problem, export the processor memory file and make your edits in the PC5 text file. Then import this text file.

Important: You cannot enter floating point numbers in scientific notation with negative exponents in complex expressions. Instead, use the decimal equivalent or put the number in a floating point file and use the data address in the complex expression.

Table 4.D
Character Lengths for Operators

This Operation:	Using this Operator:	Uses this Number of Characters:
math binary	+, -, *,	3
	OR, **	4
	AND, XOR	5
math unary	- (negate)	2
	LN *	3
	FRD, TOD, DEG*, RAD*, SQR, NOT, LOG*, SIN*, COS*, TAN*, ASN*, ACS*, ATN*	4
comparative	=, <, >	3
	<>, <=, >=	4

* Available in Enhanced PLC-5 processors only.

Determining the Order of Operation

The operations you write into the expression are performed by the processor 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 processor to perform the operation within the parentheses ahead of other operations.

Operations of equal order are performed left to right. The expression you use must include an operator. Table 4.E shows the order of operation.

Table 4.E
Order of Operation for CPT Expressions

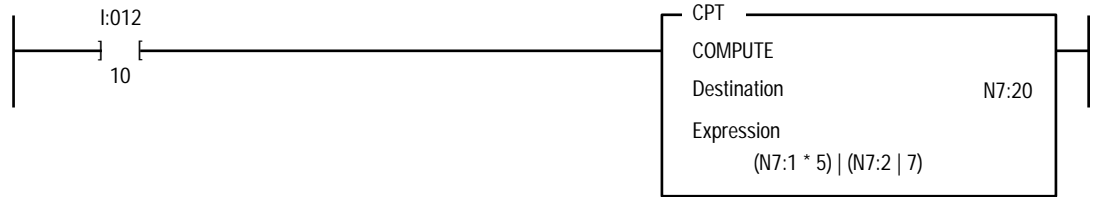
Order	Operation	Description
1	**	exponential (X^Y) Enhanced PLC-5 processors only
2	-	negate
	NOT	bitwise complement
3	*	multiply
		divide
4	+	add
	-	subtract
5	AND	bitwise AND
6	XOR	bitwise exclusive OR
7	OR	bitwise OR

Expression Examples

Single Value: The expression SQR (N7:4) with destination N7:20 tells the processor to take the square root of the value stored at N7:4 and store the result at N7:20.

Multiple Values: With Enhanced PLC-5 processors, you can also use functions to operate on one or more values in the expression (complex expressions) for compute and compare operations. Complex expressions can be up to 80 characters long (spaces and parentheses are considered characters). For example, you could enter an expression such as:

Example:



If the input word 12, bit 10 is set, multiply the value of N7:1 by 5. Divide this result by the quotient of N7:2 divided by 7. If N7:1=5 and N7:2=9, the result is 25. (The result is rounded to the nearest whole number because the constants 5 and 7 were specified as whole numbers.)

When you use complex expressions, if any operand is floating point, the entire expression is evaluated as floating point:

Example:



If the input word 12, bit 10 is set, multiply the value of N7:1 by 5. Divide this result by the quotient of N7:2 divided by 7. If N7:1=5 and N7:2=9, the result is 19. (The result is rounded differently because the constants 5.0 and 7.0 were specified to 1 decimal place.)

Entering the Destination

Enter a direct or indirect logical address for the destination. The instruction stores the result of the operation in the destination address.

Important: The processor automatically converts the data type specified by the source address to that specified by the destination address. The processor uses BCD for display or compatibility with PLC-2 family processors. You must program any BCD conversions.

Using CPT Functions

Use functions to operate on one or more values in the expression of a CPT instruction to perform these types of operations:

- convert from one number format to another
- manipulate numbers
- perform trigonometric functions

The instruction performs the function(s) you specify based on a mnemonic. When you enter the expression, enter the mnemonic as the prefix to the address of the value on which you want to operate, or as a prefix to the value itself when entered as a program constant.

Important: Floating-point numbers are 32-bit values. Integers are 16-bit values. The instruction automatically converts the data types found in the expression to the data type specified by the destination address.



ATTENTION: If the expression or destination addresses require conversion from 32-bit to 16-bit data and the value is too large, the processor sets an overflow bit in S:0/1 and sets a minor fault (S10:14). The resulting erroneous value could lead to a dangerous situation. Monitor this bit in your ladder program.

Table 4.F lists the CPT functions you can use.

Table 4.F
CPT Functions for Number Conversion

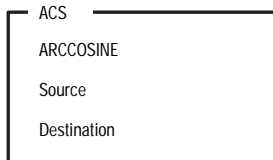
Mnemonic	Title	Description
RAD *	radians	Converts from degrees to radians
DEG *	degrees	Converts from radians to degrees
TOD	to BCD	Converts from integer to BCD (supports 4-digit BCD numbers)
FRD	from BCD	Converts from BCD to integer (supports 4-digit BCD numbers)
SQR	square root	Takes the square root of the number; accurate to 6 significant digits
LOG *	–	Log to the base 10; accurate to 6 significant digits
LN *	–	Natural log; accurate to 6 significant digits
SIN *	sine; manipulated in radians, accurate to 6 significant digits	
COS *	cosine; manipulated in radians, accurate to 6 significant digits	
TAN *	tangent; manipulated in radians, accurate to 6 significant digits	
ASN *	inverse sine; manipulated in radians, accurate to 6 significant digits	
ACS *	inverse cosine; manipulated in radians, accurate to 6 significant digits	
ATN *	inverse tangent; manipulated in radians, accurate to 6 significant digits	

* Available in Enhanced PLC-5 processors only.

You can use the above CPT arithmetic functions within expressions or as stand-alone instructions; see the individual instructions described in this chapter.

Arc Cosine (ACS) (Enhanced PLC-5 Processors Only)

Description:



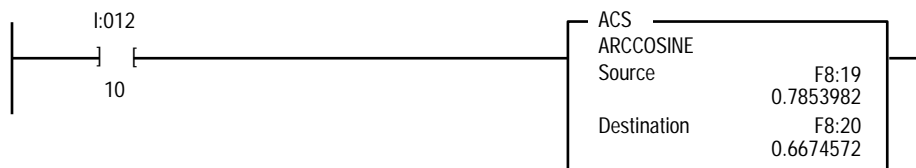
Use the ACS instruction to take the arc cosine of the source (in radians) and store the result (in radians) in the Destination. See Table 4.G for status flags for the ACS instruction.

The Source must be greater than or equal to -1 and less than or equal to 1 . If it is not in this range, the processor returns a !NAN! result in the Destination. The resulting value in the Destination is always greater than or equal to 0 and less than or equal to π (where $\pi = 3.141592$).

Table 4.G
Updating Arithmetic Status Flags for an ACS Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	always resets

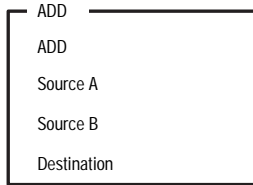
Example:



If input word 12, bit 10 is set, take the arc cosine of the value in F8:19 and store the result in F8:20.

Addition (ADD)

Description:



Use the ADD instruction to add one value (Source A) to another value (Source B) and place the result in the destination. Source A and Source B can either be values or addresses that contain values. See Table 4.H for status flags for the ADD instruction.

Important: The ADD instruction executes once each scan as long as the rung is true; if you only want values added once, include the ONS command (see chapter 13).

Table 4.H
Updating Arithmetic Status Flags for an ADD Instruction

With this Bit:	The Processor:
Carry (C)	sets if carry generated; otherwise resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

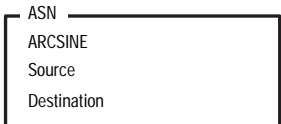
Example:



If input word 12, bit 10 is set, add the value in N7:3 to the value in N7:4 and store the result in N7:20.

Arc Sine (ASN) (Enhanced PLC-5 Processors Only)

Description:



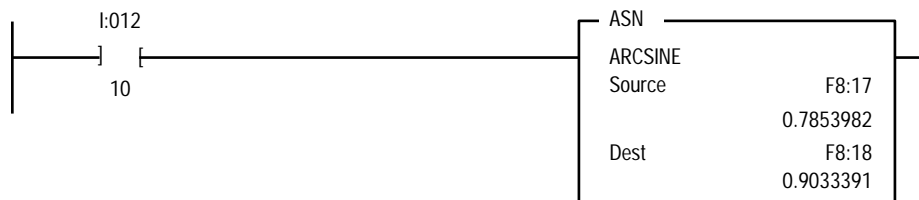
Use the ASN instruction to take the arc sine the source (in radians) and store the result (in radians) in the Destination. See Table 4.I for status flags for ASN instruction.

The Source must be greater than or equal to -1 and less than or equal to 1 . If it is not in this range, the processor returns a !NAN! result in the Destination. The resulting value in the Destination is always greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$ (where $\pi = 3.141592$).

Table 4.I
Updating Arithmetic Status Flags for an ASN Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	always resets

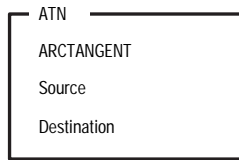
Example:



If input word 12, bit 10 is set, take the arc sine of the value in F8:17 and store the result in F8:18.

Arc Tangent (ATN) (Enhanced PLC-5 Processors Only)

Description:



Use the ATN instruction to take the arc tangent of the source (in radians) and store the result (in radians) in the Destination. The resulting value in the Destination is always greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$ (where $\pi = 3.141592$). See Table 4.J for status flags for ATN instruction.

Table 4.J
Updating Arithmetic Status Flags for an ATN Instruction

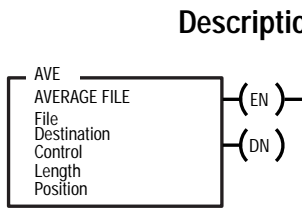
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



If input word 12, bit 10 is set, take the arc tangent of the value in F8:21 and store the result in F8:22.

Average File (AVE) (Enhanced PLC-5 Processors Only)



Description:

The AVE instruction calculates the average of a set of values. When the rung goes from false to true, the value at the current position is added to the next value, which is added to the next value, and so on. See Table 4.K for status flags for AVE instruction.

Each time another value is added, the position field and the status word (S:24) is incremented. The final sum is divided by the number of values added and the result is stored in the destination.

Table 4.K
Updating Arithmetic Status Flags for an AVE Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

An overflow can occur if:

- the intermediate sum exceeds the maximum floating point value
- the destination is an integer address and the final value is greater than 32,767 or less than -32,768

If an overflow occurs, the processor stops the calculation, sets the .ER bit, and the Destination remains unchanged. The position identifies the element that caused the overflow. When you clear the .ER bit, the position resets to 0 and the average is recalculated.

Important: Use the RES instruction to clear the status flags.

Entering Parameters

To program the AVE instruction, you must provide the processor with the following:

- **File** is the address that contains the first value to be added. This address can be floating point or integer.
- **Destination** is the address where the result of the instruction is stored. This address can be floating point or integer.
- **Control** is the address of the control structure in the control area (R) of processor memory. The processor stores information such as the length, position and status, and uses this information to execute the instruction.
- **Length** is the number of words in the file (1-1000).
- **Position** points to the word that the instruction is currently using.

Using Status Bits

To use the AVE instruction correctly, examine status bits in the control structure. Address these bits by mnemonic.

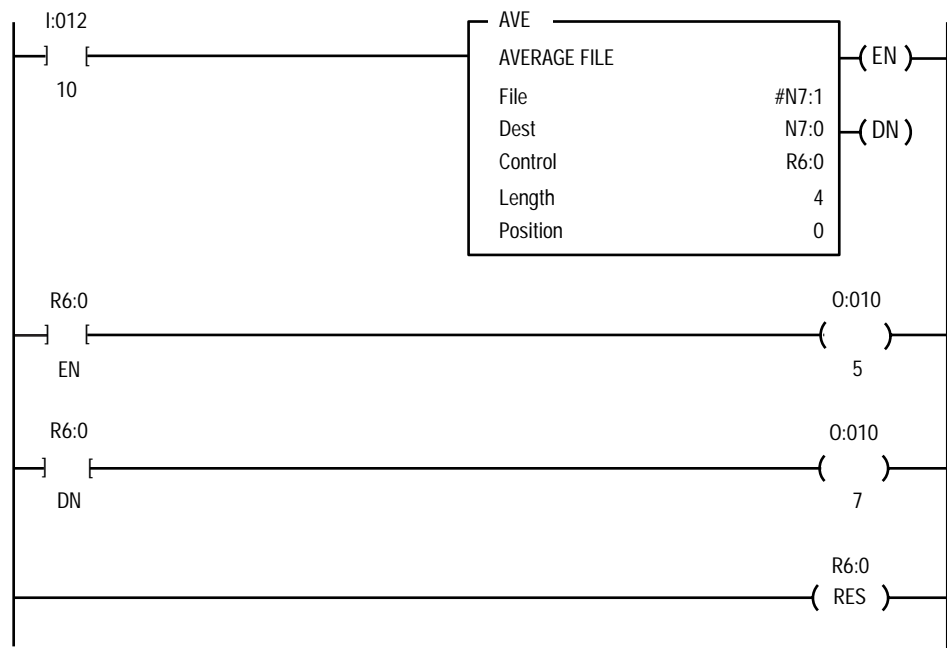
This Bit:	Is Set:
Enable .EN (bit 15)	on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition.
Done .DN (bit 13)	after the instruction finishes operating. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the operation generates an overflow. The instruction stops until the ladder program resets the .ER bit.

Important: The AVE instruction calculates the average using floating point regardless of the type specified for the file or destination parameters.



ATTENTION: The AVE instruction increments the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise, unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

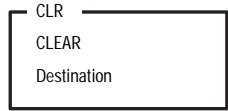
Example:



If input word 12, bit 10 is set, the AVE instruction is enabled. The values in N7:1, N7:2, N7:3, and N7:4 are added together and divided by 4. The result is stored in N7:0. When the calculation is complete, output word 10, bit 7 is set. Then the RES instruction resets the status bits of the control file R6:0.

Clear (CLR)

Description:

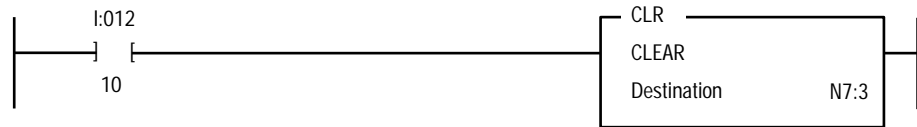


Use the CLR instruction to set all the bits of a word to zero. The destination must be a word address. See Table 4.L for status flags for CLR instruction.

Table 4.L
Updating Arithmetic Status Flags for a CLR Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	always sets
Sign (S)	always resets

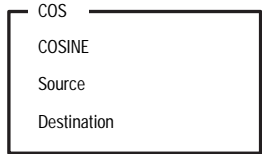
Example:



If input word 12, bit 10 is set, clear all of the bits in N7:3 to zero.

Cosine (COS) (Enhanced PLC-5 Processors Only)

Description:



Use the COS instruction to take the cosine of a number (Source in radians) and store the result in the Destination. See Table 4.M for status flags for COS instruction.

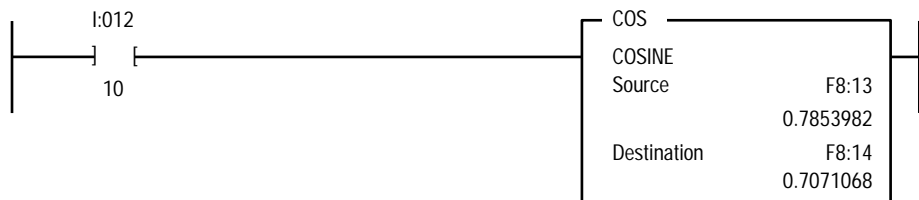
The Source must be greater than or equal to -205887.4 and less than or equal to 205887.4 . If it is not in this range, the processor returns a !INF! result in the Destination. The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1 .

Important: For greatest accuracy, the source data should be greater than or equal to -2π and less than or equal to 2π .

Table 4.M
Updating Arithmetic Status Flags for an COS Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

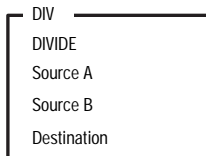
Example:



If input word 12, bit 10 is set, take the cosine of the value in F8:13 and store the result in F8:14.

Divide (DIV)

Description:



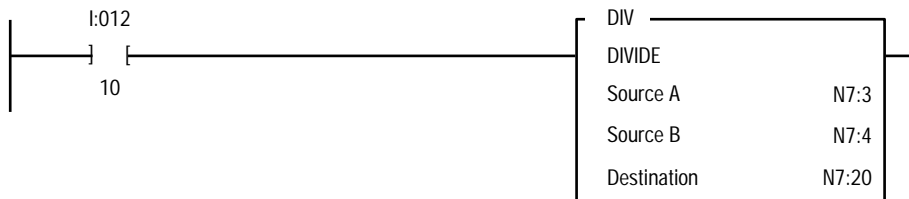
Use the DIV instruction to divide one value (Source A) by another value (Source B) and place the result in the Destination. Source A and Source B can either be values or addresses that contain values. See Table 4.N for status flags for DIV instruction.

Important: The compute instructions execute for each scan as long as the rung is true; if you only want values computed once, include the ONS command (see chapter 13).

Table 4.N
Updating Arithmetic Status Flags for a DIV Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if division by zero or if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets; undefined if overflow is set
Sign (S)	sets if result is negative; otherwise resets; undefined if overflow is set

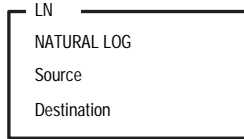
Example:



If input word 12, bit 10 is set, divide the value in N7:3 by the value in N7:4 and store the result in N7:20.

Natural Log (LN) (Enhanced PLC-5 Processors Only)

Description:



Use the LN instruction to take the natural log of the value in the Source and store the result in the Destination. See Table 4.O for status flags for LN instruction.

If the Source is equal to 0, the result in the Destination will be ! - INF ! ; if the value in the Source is less than 0, the result in the Destination will be ! NAN ! . The resulting value in the Destination is always greater than or equal to -87.33655 and less than or equal to 88.72284.

Table 4.0
Updating Arithmetic Status Flags for an LN Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

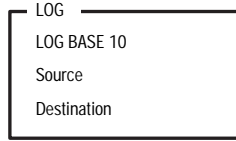
Example:



If input word 12, bit 10 is set, take the natural log of the value in N7:0 and store the result in F8:20.

Log to the Base 10 (LOG) (Enhanced PLC-5 Processors Only)

Description:



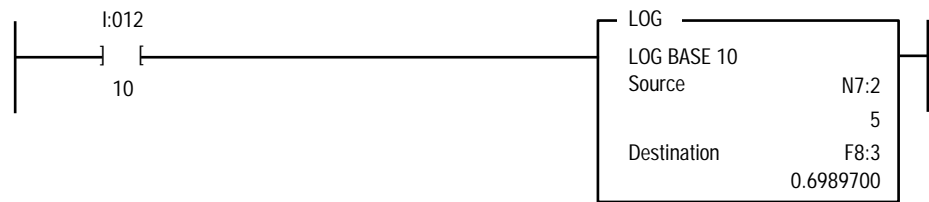
Use the LOG instruction to take the log base 10 of the value in the Source and store the result in the Destination. See Table 4.P for status flags for LOG instruction.

If the Source is equal to 0, the result in the Destination will be ! -INF ! ; if the value in the Source is less than 0, the result in the Destination will be !NAN! . The resulting value in the Destination is always greater than or equal to -37.92978 and less than or equal to 38.53184.

Table 4.P
Updating Arithmetic Status Flags for an LOG Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



If input word 12, bit 10 is set, take the log base 10 of the value in N7:2 and store the result in F8:3.

Multiply (MUL)

Description:

Use the MUL instruction to multiply one value (Source A) by another value (Source B) and place the result in the destination. Source A and Source B can be values or addresses. See Table 4.Q for status flags for MUL instruction.

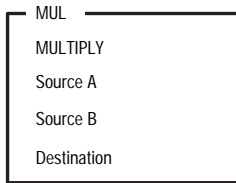
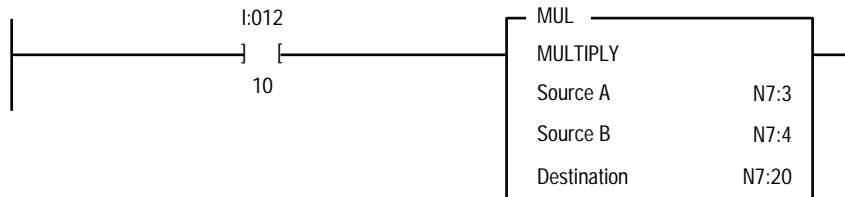


Table 4.Q
Updating Arithmetic Status Flags for a MUL Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

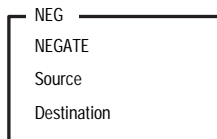
Example:



If input word 12, bit 10 is set, multiply the value in N7:3 by the value in N7:4 and store the result in N7:20.

Negate (NEG)

Description: Use the NEG instruction to change the sign of a value. If you negate a negative value, the result is positive; if you negate a positive value, the result is negative. See Table 4.R for status flags for NEG instruction.

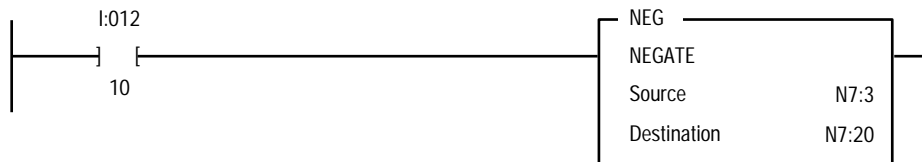


Important: The compute instructions execute for each scan as long as the rung is true; if you only want values computed once, include the ONS command (see chapter 13).

Table 4.R
Updating Arithmetic Status Flags for a NEG Instruction

With this Bit:	The Processor:
Carry (C)	sets if the operation generates a carry
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

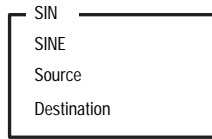
Example:



If input word 12, bit 10 is set, take the opposite sign of the value in N7:3 and store the result in N7:20.

Sine (SIN) (Enhanced PLC-5 Processors Only)

Description:



Use the SIN instruction to take the sine of a number (Source in radians) and store the result in the Destination. See Table 4.S for status flags for SIN instruction.

The Source must be greater than or equal to -205887.4 and less than or equal to 205887.4 . If it is not in this range, the processor returns a !INF! result in the Destination. The resulting value in the Destination is always greater than or equal to -1 and less than or equal to 1 .

Important: For greatest accuracy, the source data should be greater than or equal to -2π and less than or equal to 2π .

Table 4.S
Updating Arithmetic Status Flags for an SIN Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

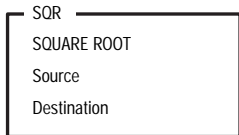
Example:



If input word 12, bit 10 is set, take the sine of F8:11 and store the result in F8:12.

Square Root (SQR)

Description:



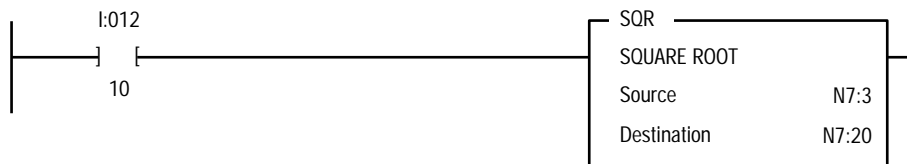
Use the SQR instruction to take the square root of a value and store the result in the destination. The source can be a value or an address. If the source value is negative, the processor takes its absolute value and performs the square root function. See Table 4.T for status flags for SQR instruction.

Important: The SQR instruction executes once for each scan as long as the rung is true; if you only want values computed once, include the ONS command (see chapter 13).

Table 4.T
Updating Arithmetic Status Flags for a SQR Instruction

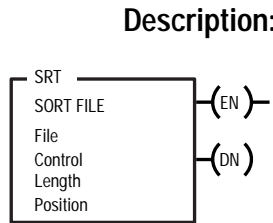
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated during floating-point to integer conversion; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	always resets

Example:



If input word 12, bit 10 is set, take the square root of the value in N7:3 and store the result in N7:20.

Sort File (SRT) (Enhanced PLC-5 Processors Only)



The SRT instruction sorts a set of values into ascending order. This instruction is executed on a false-to-true transition.

Important: Make sure the file length value you specify in the instruction does not cause the indexed address to exceed the file bounds. The processor does not check this unless you exceed the data file area of memory. If the indexed address exceeds the data file area, the processor initiates a run-time error and sets a major fault. The processor does not check to see whether the indexed address crosses file types, such as N7 to F8.

Entering Parameters

To program the SRT instruction, you must provide the processor with the following:

Parameter:	Definition:
file	the address that contains the first value to be sorted. This address can be floating point or integer.
control	the address of the control structure in the control area (R) of processor memory. The processor stores information such as the length, position and status, and uses this information to execute the instruction.
length	the number of words in the file (1-1000).
position	points to the element that the instruction is currently using.

Using Status Bits

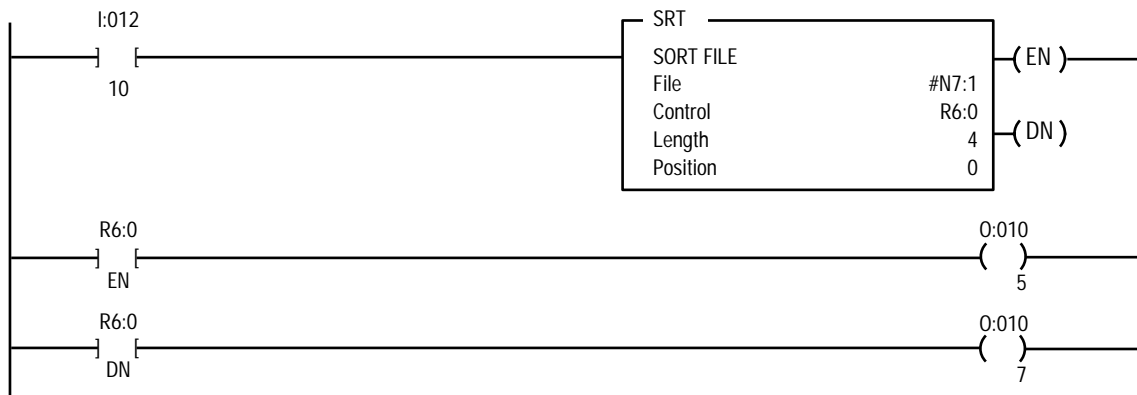
To use the SRT instruction correctly, the ladder program must examine status bits in the control structure. You address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition.
Done .DN (bit 13)	after the instruction finishes operating. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the length value is less than or equal to zero or when the position value is less than zero.



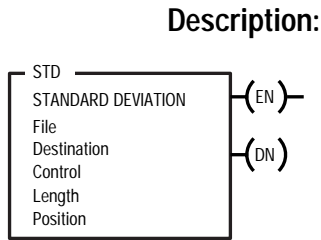
ATTENTION: The SRT instruction manipulates the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise, unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

Example:



If input word 12, bit 10 is set, the SRT instruction is enabled. The elements N7:1, N7:2, N7:3, and N7:4 are sorted into ascending order. When the sort operation is complete, output word 10, bit 7 is set.

Standard Deviation (STD) (Enhanced PLC-5 Processors Only)



The STD instruction calculates the standard deviation of a set of values and stores the result in the destination. This instruction is executed on a false-to-true transition. See Table 4.U for status flags for STD instruction.

The standard deviation is calculated according to the following formula:

$$\text{Standard Deviation} = \sqrt{\left(\frac{\text{SUM}((x_i - \text{AVE}(x_i))^2)}{(N - 1)} \right)}$$

Where:

- SUM() – summation function of the enclosed variables
- AVE () – average function of the enclosed variables
- xi – variable elements of the data file
- N – number of elements in the data file

Important: Make sure the file length value you specify in the instruction does not cause the indexed address to exceed the file bounds. The processor does not check this unless you use an indexed indirect address or exceed the data file area of memory. If the indexed address exceeds the data file area, the processor initiates a run-time error and sets a major fault. The processor does not check to see whether the indexed address crosses file types, such as N7 to F8.

Table 4.U
Updating Arithmetic Status Flags for an STD Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	always resets

There are two ways an overflow can occur:

- the intermediate sum exceeds the maximum floating point value (floating point values are: $\pm 1.1754944 \times 10^{-38}$ to $\pm 3.4028237 \times 10^{38}$)
- the destination is an integer address and the final value is greater than 32,767

If an overflow occurs, the processor stops the calculation, sets the .ER bit, and leaves the destination unchanged. The position identifies the element that caused the overflow. When you clear the .ER bit, the position resets to 0 and the standard deviation is recalculated.

Important: Use the RES instruction to clear the status bits.

Entering Parameters

To program the STD instruction, you must provide the processor with the following:

Parameter:	Defines:
file	the address that contains the first value to be calculated. This address can be floating point or integer.
destination	the address where the result of the instruction is stored. This address can be floating point or integer.
control	the address of the control structure in the control area (R) of processor memory. The processor stores information such as the length, position and status, and uses this information to execute the instruction.
length	the number of words in the file (1-1000).
position	points to the element that the instruction is currently using.

Using Status Bits

To use the STD instruction correctly, examine status bits in the control structure. You address these bits by mnemonic.

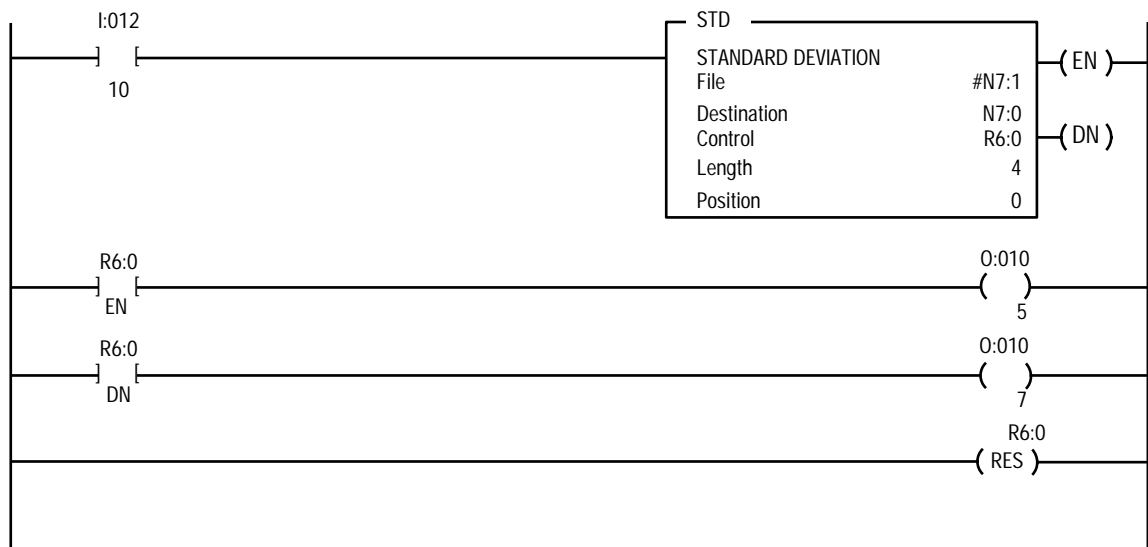
This Bit:	Is Set:
Enable .EN (bit 15)	on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition.
Done .DN (bit 13)	after the instruction finishes operating. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the operation generates an overflow. The instruction stops until the ladder program resets the .ER bit.

Important: The STD instruction calculates the standard deviation using floating point regardless of the type specified for the file or destination parameters.



ATTENTION: The STD instruction manipulates the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise, unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

Example:



If input word 12, bit 10 is set, the STD instruction is enabled. The elements N7:1, N7:2, N7:3, and N7:4 are used to calculate the standard deviation. When the calculation is complete, output word 10, bit 7 is set. Then the RES instruction resets the status bits of the control file R6:0.

Subtract (SUB)

Description:

SUB
SUBTRACT
Source A
Source B
Destination

Use the SUB instruction to subtract one value (Source B) from another value (Source A) and place the result in the destination. Source A and Source B can be values or addresses that contain values. See Table 4.V for status flags for SUB instruction.

Important: The SUB instruction executes once for each scan as long as the rung is true; if you only want values subtracted once, include the ONS command (see chapter 13).

Table 4.V
Updating Arithmetic Status Flags for a SUB Instruction

With this Bit:	The Processor:
Carry (C)	sets if borrow generated; otherwise resets
Overflow (V)	sets if underflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



If input word 12, bit 10 is set, subtract the value in N7:4 from the value in N7:3 and store the result in N7:20.

Tangent (TAN) (Enhanced PLC-5 Processors Only)

Description:



Use the TAN instruction to take the tangent of a number (Source in radians) and store the result in the Destination. See Table 4.W for status flags for TAN instruction.

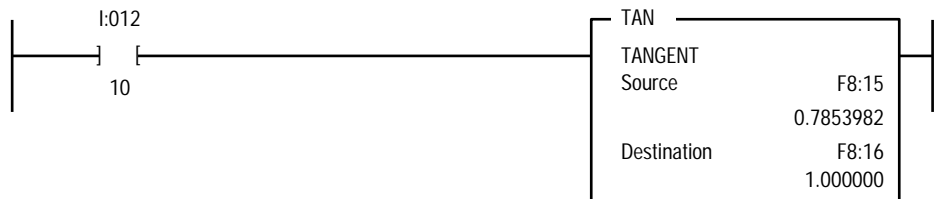
The value in the Source must be greater than or equal to -102943.7 and less than or equal to 102943.7 . If it is not in this range, the processor returns a !INF! result in the Destination. The resulting value in the Destination is always a real number.

Important: For greatest accuracy, the source data should be greater than or equal to $-\pi/2$ and less than or equal to $\pi/2$.

Table 4.W
Updating Arithmetic Status Flags for an TAN Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

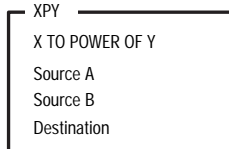
Example:



If input word 12, bit 10 is set, take the tangent of the value in F8:15 and store the result in F8:16.

X to the Power of Y (XPY) (Enhanced PLC-5 Processors Only)

Description:



Use the XPY instruction to raise a value (Source A) to a power (Source B) and store the result in the Destination. If the value in Source A is negative, the exponent (Source B) should be an integer value; if the exponent is not an integer (for example, if it is a floating point value), the overflow bit is set and the absolute value of the base is used in the calculation. See Table 4.X for status flags for XPY instruction.

The XPY instruction uses the following algorithm:

$$XPY = 10^{**} (Y * \log (X))$$

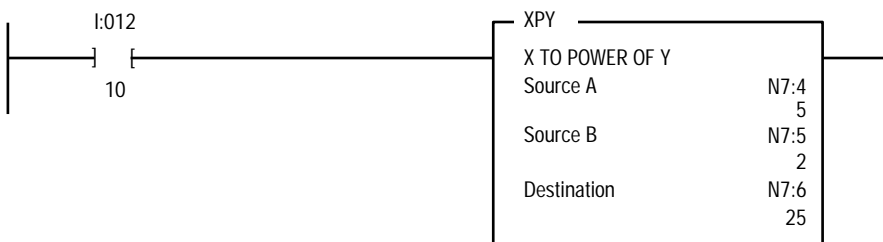
If any of the intermediate operations in this algorithm produce an overflow, the arithmetic minor fault bit is set (S:10/14). The arithmetic status flag bit is set only if the final result is an overflow.

Important: Keep in mind that x0 is equal to 1; 0x is equal to 0. For floating point numbers, 00 is equal to !NAN! (an invalid mathematical value) and for integers, 00 is equal to -1.

Table 4.X
Updating Arithmetic Status Flags for an XPY Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



If input word 12, bit 10 is set, take the value in N7:4, raise it to the power of the value in N7:5, and stores the result in N7:6.

Notes:

Logical Instructions AND, NOT, OR, XOR

Using Logical Instructions

These instructions (Table 5.A) perform logical operations.

Table 5.A
Available Logical Instructions

If You Want to:	Use this Instruction:	Found on Page:
Perform an AND operation	AND	5-2
Perform a NOT operation	NOT	5-3
Perform an OR operation	OR	5-4
Perform an XOR operation	XOR	5-5

The parameters you enter are program constants or direct logical addresses.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Arithmetic Status Flags

The arithmetic status bits are in word 0 bits 0-3 in the processor status file (S). Table 5.B lists the status flags:

Table 5.B
Arithmetic Status Flags

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

AND Operation (AND)

Description: Use the AND instruction to perform an AND operation using the bits in the two source addresses.

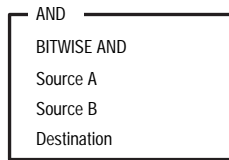


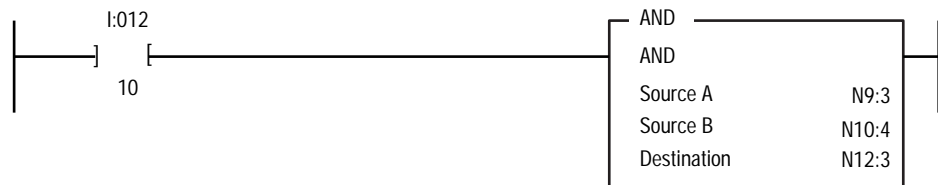
Table 5.C
Truth Table for an AND Operation

Source A	Source B	Result
0	0	0
1	0	0
0	1	0
1	1	1

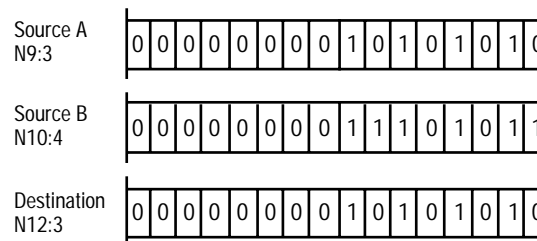
Table 5.D
Updating Arithmetic Status Flags for an AND Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

Example:



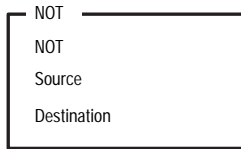
If input word 12, bit 10 is set, the processor performs an AND operation on N9:3 and N10:4 and stores the result in N12:3.



NOT Operation (NOT)

Description:

Use the NOT instruction to perform a NOT operation using the bits in the source address. This operation is also known as a bit inversion.



Important: The NOT instruction is not available on PLC-5/15 series A processors.

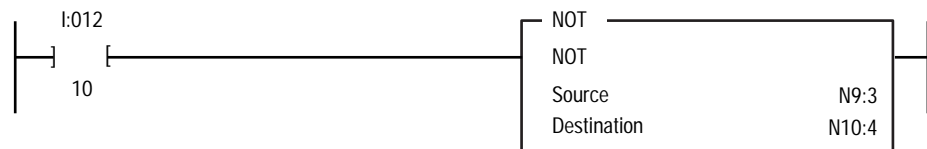
Table 5.E
Truth Table for a NOT Operation

Source	Result
0	1
1	0

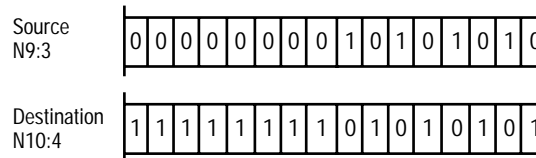
Table 5.F
Updating Arithmetic Status Flags for a NOT Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

Example:



If input word 12, bit 10 is set, the processor performs a NOT operation on N9:3 and stores the result in N10:4



OR Operation (OR)

Description: Use the OR instruction to perform an OR operation using the bits in the two sources (constants or addresses).

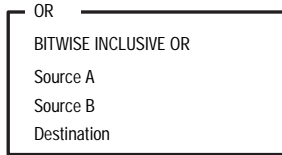


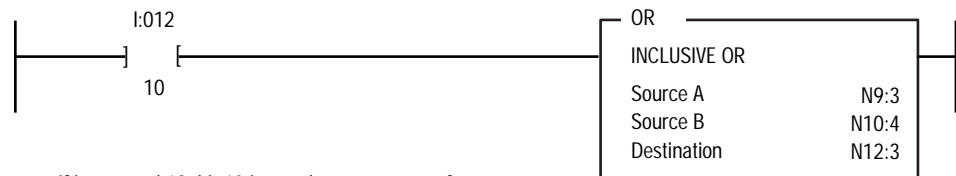
Table 5.G
Truth Table for an OR Operation

Source A	Source B	Result
0	0	0
1	0	1
0	1	1
1	1	1

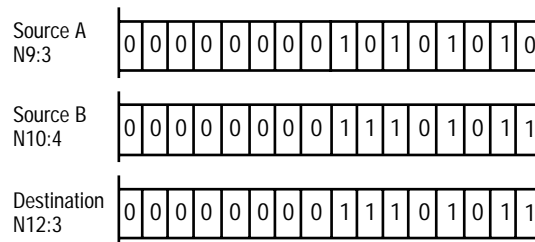
Table 5.H
Updating Arithmetic Status Flags for an OR Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

Example:



If input word 12, bit 10 is set, the processor performs an OR operation on N9:3 and N10:4 and stores the result in N12:3.



Exclusive OR Operation (XOR)

Description: Use the XOR instruction to perform an exclusive OR operation using the bits in the two sources (constants or addresses).

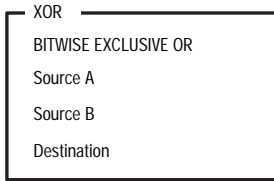


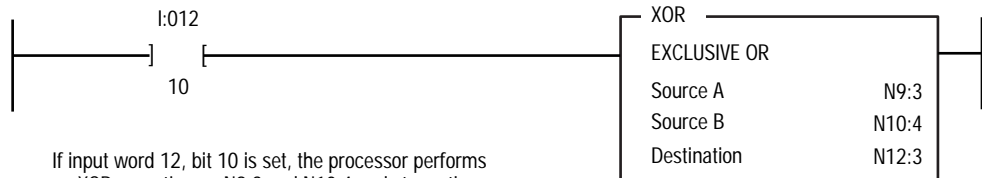
Table 5.I
Truth Table for an XOR Operation

Source A	Source B	Result
0	0	0
1	0	1
0	1	1
1	1	0

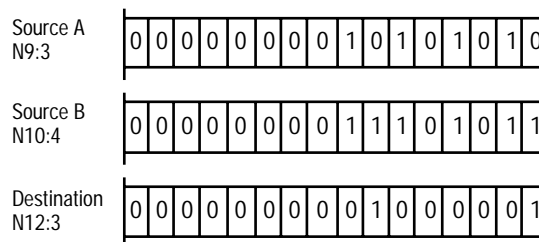
Table 5.J
Updating Arithmetic Status Bits for an XOR Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if most-significant bit is set; otherwise resets

Example:



If input word 12, bit 10 is set, the processor performs an XOR operation on N9:3 and N10:4 and stores the result in N12:3.



Notes:

Conversion Instructions FRD and TOD, DEG and RAD

Using the Conversion Instructions

The conversion instructions convert integer to BCD and convert BCD to integer (using TOD and FRD). For example, use TOD and FRD for signals to/from BCD I/O devices, for display purposes, or for number compatibility with PLC-2 family processors. You can also convert radians to degrees and degrees to radians (using DEG and RAD). For example, you can use DEG and RAD with the trigonometric instructions (see chapter 4).

Table 6.A lists the available conversion instructions.

Table 6.A
Available Conversion Instructions

If You Want to:	Use this Instruction:	Found on Page:
Convert from integer to BCD	TOD	6-2
Convert from BCD to integer	FRD	6-2
Convert radians to degrees	DEG*	6-3
Convert degrees to radians	RAD*	6-4

* These instructions are only supported by Enhanced PLC-5 processors.

The parameters you enter are program constants or logical addresses of the values you want.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Arithmetic Status Flags

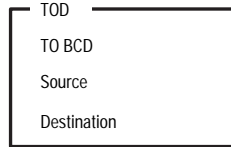
The arithmetic status flags are in word 0 bits 0-3 in the processor status file (S2). Table 6.B lists the status flags:

Table 6.B
Arithmetic Status Flags

This Bit:	Description:
S:0/0	Carry (C)
S:0/1	Overflow (V)
S:0/2	Zero (Z)
S:0/3	Sign (S)

Convert to BCD (TOD)

Description:

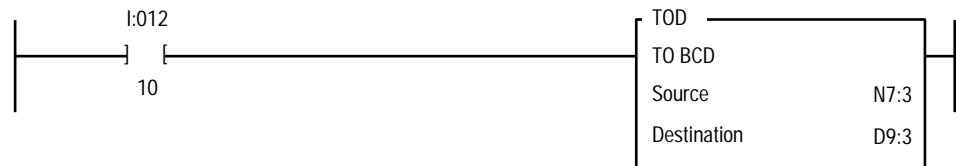


Use the TOD instruction to convert an integer value to a BCD value. If the integer value is greater than 9999, the processor stores 9999 and sets the overflow bit. If the integer value is negative, the processor stores 0 in the destination and sets the overflow and zero status bits.

Table 6.C
Updating Arithmetic Status Flags for a TOD Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if integer value is outside the range 0-9999; otherwise resets
Zero (Z)	sets if destination value is negative or zero; otherwise resets
Sign (S)	always resets

Example:



If input word 12, bit 10 is set, convert the value in N7:3 to a BCD value and store the result in D9:3.

Convert from BCD (FRD)

Description:



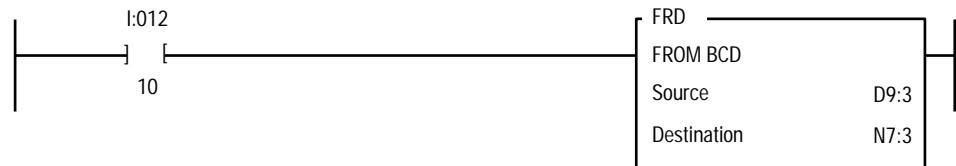
Use the FRD instruction to convert a BCD value to an integer value. Convert BCD values to integer values before you manipulate those values with ladder logic because the processor treats BCD values as integer values. The actual BCD value may be lost or distorted.

Table 6.D
Updating Arithmetic Status Flags for a TOD Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if destination value is zero; otherwise resets
Sign (S)	always resets

The FRD instruction will convert a non-decimal number without any error condition. For example, if a “C” is in the source, it is converted to “12,” even though “C” is not a valid decimal number.

Example:



If input word 12, bit 10 is set, convert the value in D9:3 to an integer value and store the result in N7:3.

**Degree (DEG)
(Enhanced PLC-5 Processors Only)**

Description:



Use the DEG instruction to convert radians (Source) to degrees and store the result in the Destination (Source times $180/\pi$).

**Table 6.E
Updating Arithmetic Status Flags for an DEG Instruction**

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



If input word 12, bit 10 is set, convert the value in F8:7 to degrees and store the result on F8:8.

Radian (RAD) (Enhanced PLC-5 Processors Only)

Description:

Use the RAD instruction to convert degrees (Source) to radians and store the result in the Destination (Source times $\pi/180$).

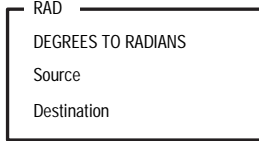


Table 6.F
Updating Arithmetic Status Flags for an RAD Instruction

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Example:



If input word 12, bit 10 is set, convert the value in N7:9 to radians and store the result on F8:10.

Bit Modify and Move Instructions

BTD, MOV, MVM

Using Bit Modify and Move Instructions

The bit modify and move instructions let you modify and move bits. Table 7.A lists the available move instructions.

Table 7.A
Available Bit Modify and Move Instructions

If You Want to:	Use this Instruction:	Found on Page:
Move bits within a word or between words	BTD	7-2
Copy the value in one word to another word	MOV	7-3
Copy the desired part of a 16-bit value by masking the rest of the value with a mask	MVM	7-4

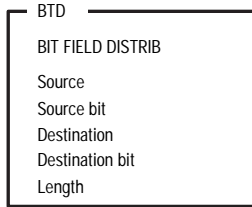
These instructions operate on 16-bit integer, binary or floating point numbers to move or copy bits between words. The MVM instruction uses a mask to either pass or block source data bits. A mask passes data when the mask bits are set (1); a mask blocks data when the mask bits are reset (0). The mask must be the same word size as the source and destination.

When rounding floating point numbers during a move to an integer word, the processor does not correctly round numbers less than -1 .

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Bit Distribute (BTD)

Description:



The BTD instruction is an output instruction that moves up to 16 bits of data within a word or between words. The Source remains unchanged. The instruction writes over the Destination with the specified bits. If the length of the bit field extends beyond the Destination word, the processor does not save the overflow bits. These overflow bits are lost; they do not wrap into the next word.

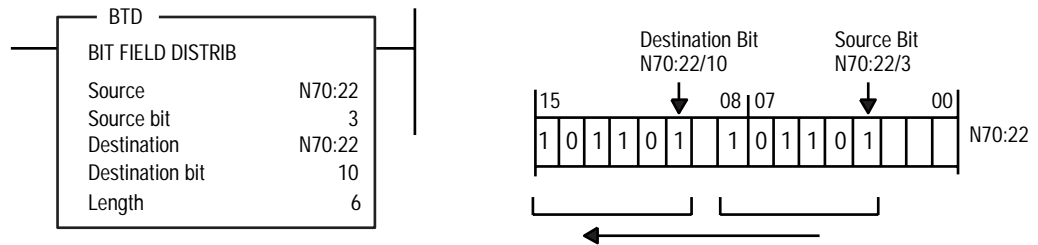
On each scan, when the rung that contains the BTD instruction is true, the processor moves the bit field from the source word to the destination word. To move data within a word, enter the same address for the source and destination.

Entering Parameters

To program the BTD instruction, you must provide the processor with the following:

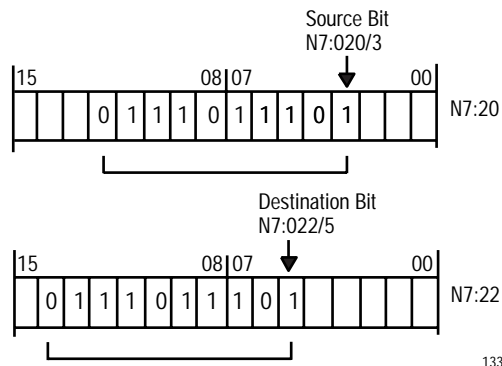
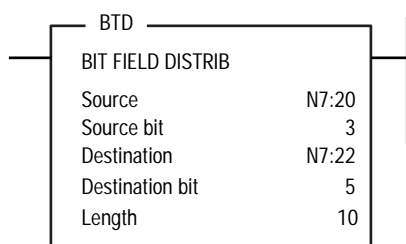
Parameter:	Definition:
Source	the address of the source word in binary or integer. The source remains unchanged.
Source bit	the number of the bit (lowest bit number) in the source word from which to start the move.
Destination	the address of the destination word in a binary or integer file. The instruction writes over any data already stored at the destination.
Destination bit	the number of the bit (lowest bit number) in the destination word where the processor starts copying the bits from the source word.
Length	the number of bits to be moved.

Example: Moving Bits Within a Word



13384

**Example:
Moving Bits Between Words**



13384

Important: Bits are lost if they extend beyond the end of the destination word; the bits are **not** wrapped to the next higher word.

Move (MOV)

Description:



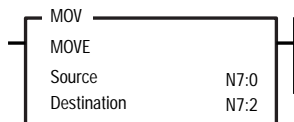
The MOV instruction is an output instruction that copies the Source address to a Destination. As long as the rung remains true, the instruction moves the data each scan.

Table 7.B describes how the processor updates the arithmetic status flags.

**Table 7.B
Updating Arithmetic Status Flags for a MOV Instruction**

With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	sets if overflow generated during floating point-to-integer conversion; otherwise resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

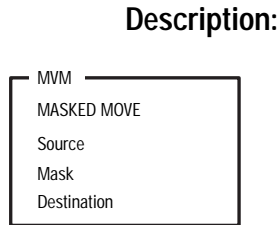
Example:



To program this instruction, you must provide the processor with the following:

Parameter:	Definition:
source	is a program constant or data address from which the instruction reads an image of the value. You can also use a symbol, as long as the symbol name is more than 1 character. The source remains unchanged.
destination	the data address to which the instruction writes the result of the operation. The instruction writes over any data stored at the destination.

Masked Move (MVM)



The MVM instruction is an output instruction that copies the Source to a Destination, and allows portions of the data to be masked. As long as the rung remains true, the instruction moves data each scan.

You can use the MVM instruction to copy I/O image, binary, or integer values. For example, use MVM to extract bit data such as status or control bits from an element that contains bit and word data.

Table 7.C describes how the processor updates the arithmetic status flags.

Table 7.C
Updating Arithmetic Status Flags for a MVM Instruction

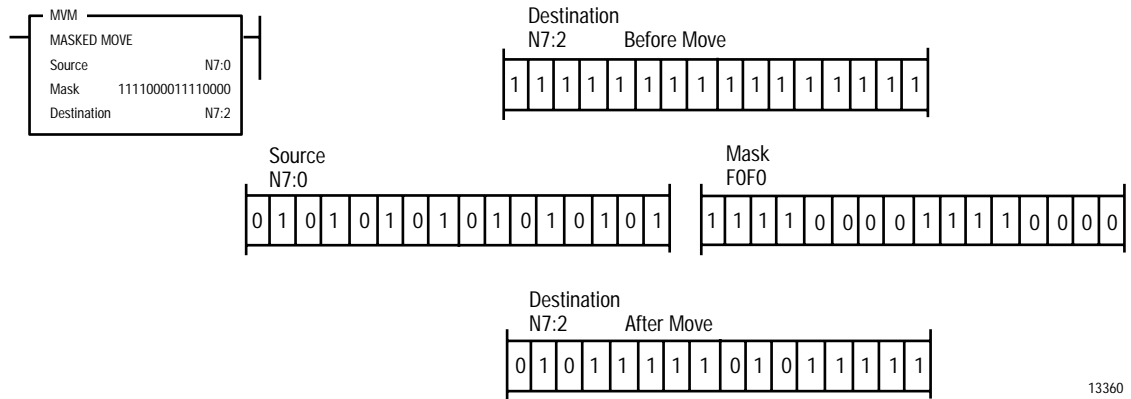
With this Bit:	The Processor:
Carry (C)	always resets
Overflow (V)	always resets
Zero (Z)	sets if result is zero; otherwise resets
Sign (S)	sets if result is negative; otherwise resets

Entering Parameters

To program this instruction, you must provide the processor with the following:

Parameter:	Definition:
Source	a program constant or data address from which the instruction reads an image of the value. The source remains unchanged.
Mask	<p>an address or hexadecimal value that specifies which bits to pass or block.</p> <p>You must set (1) mask bits to move data. Moved data overwrites destination data. Bits at the destination that correspond to zeros in the mask are not altered.</p> <p>If you want the ladder program to change the mask value, store the mask at a data address. When you enter a value in this field, make sure that you include the data type, file number and word number. For example, type B100:0.</p> <p>Otherwise, enter a hexadecimal value for a constant mask value. For example, type F800.</p>
Destination	the data address to which the instruction writes the result of the operation. The instruction writes over any data stored at the destination.

Example:



13360

Notes:

File Instruction Concepts

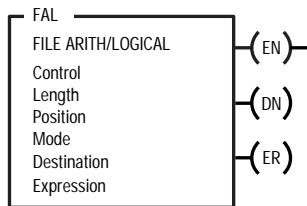
Concepts of File Operation

This chapter presents concepts of block operation for the File Arithmetic and Logical (FAL) and File Search and Compare (FSC) instructions.

The FAL instruction performs arithmetic and logical operations on blocks of words. The FSC instruction performs comparison operations on blocks of words. For specific information about the FAL or FSC instruction, see chapter 9.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Entering Parameters



You need to provide the processor with the following information when you enter a file instruction:

Parameter:	Definition:
Control	the address of the control structure in a control type (R) file. The processor uses this information to run the instruction. See "Using the Control Structure" on page 8-2.
Length	the number of words in the data block on which the file instruction operates. Enter any decimal number 1-1000.
Position	the current word within the data block that the processor is accessing. You generally enter a zero to start at the beginning of a block.
Mode	<p>the number of file words operated on each time the rung is scanned in the program. The mode lets you distribute operation on the complete block of words. Specify one of the following:</p> <ul style="list-style-type: none"> • for All mode, type an A • for Numerical mode, type a decimal number (1-1000) • for Incremental mode, type an I <p>For more information about the different modes, see "Choosing Modes of Block Operation" on page 8-5.</p>
Destination	the address where the processor stores the result of the operation. The instruction converts to the data type specified by the destination address.
Expression	<p>contains addresses, program constants, and operators that specify the source of data and the operations to be performed.</p> <p>If you enter the index prefix (#) for a destination or expression address, the processor accepts it as the address of the first word of a block to be operated upon. The processor assigns and uses the offset value in module status to process the block address. If you omit the # prefix, the processor accepts this as the address of a single work to be operated upon.</p>

Important: Make sure the index value (positive or negative) does not cause the indexed address to exceed the file type boundary. The processor does not check this unless you use an indexed indirect address or exceed the data table area of memory. If the indexed address exceeds the data table area, the processor initiates a run-time error and sets a major fault. The processor does not check to see whether the indexed address crosses file types, such as N7 to F8.



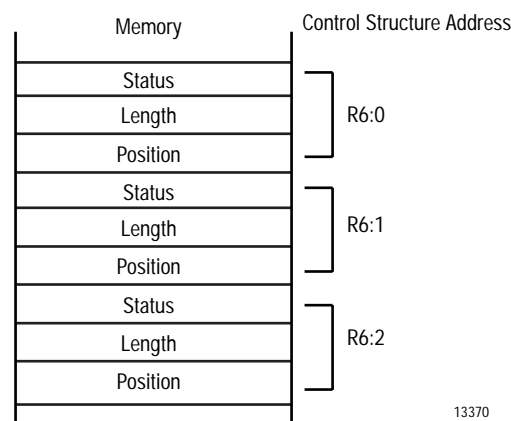
ATTENTION: Instructions with a # sign in an address manipulate the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

For more information on indexed addressing, see the chapter on addressing data table files in your software user manual.

Using the Control Structure

The control structure (file type R) controls the operation of the file instruction. Similar to a counter, it controls the file by length, position and status and control bits (Figure 8.1). You enter the control structure address (for example R6:0) in the Control field when you program a FAL or FSC instruction.

Figure 8.1
Example Control File R6:0



ATTENTION: Do not use the same control address for more than one instruction. Duplication of a control address could result in unpredictable operation, possibly causing damage to equipment and/or injury to personnel.

The control structure stores the following information:

- Status bits
- Length (.LEN) of the block (1-1000 words)
- Position (.POS) of the words that the processor is operating on

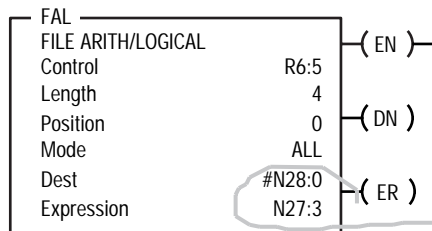
The FAL instruction and FSC instruction each has its own set of status bits. See chapter 9 for the FAL or FSC instruction for a description of these status bits.

Manipulating File Data

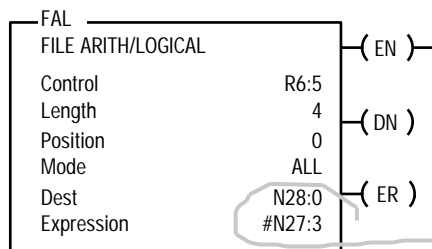
Typical data manipulations with file instructions include:

- Copying data from a
 - source word to a destination block
 - source block to a destination block
 - source block to a destination word
- Operating on data from multiple sources such as
 - source words
 - source blocks
- Storing the result in a
 - destination block
 - destination word

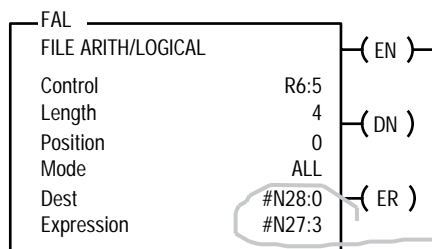
The # prefix for a destination or expression address establishes it as the address of the first word of a block to be operated upon. The absence of the # prefix establishes it as the address of a single word to be operated upon.



The # prefix for the destination address and the absence of a # prefix for the expression address establish this as a **word-to-block** operation.



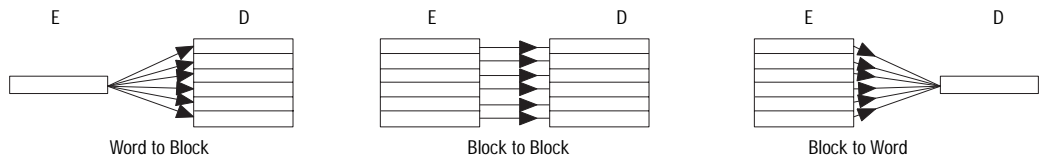
The absence of a # prefix for the destination address and the # prefix for the expression address establish this as a **block-to-word** operation.



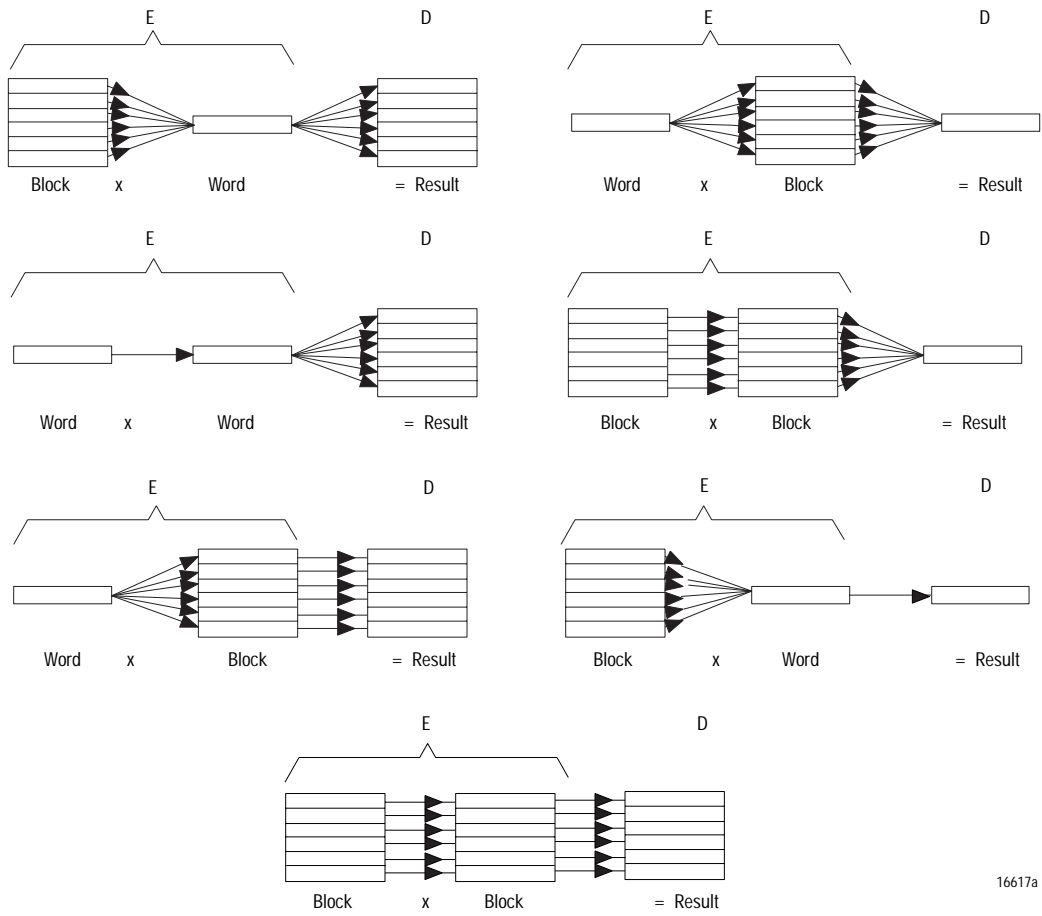
The # prefix for the destination address and the # prefix for the expression address establish this as a **block-to-block** operation.

The following example shows generic data manipulations used with file instructions (E = expression, D = destination, x = operation).

Moving Data



Operating on Data



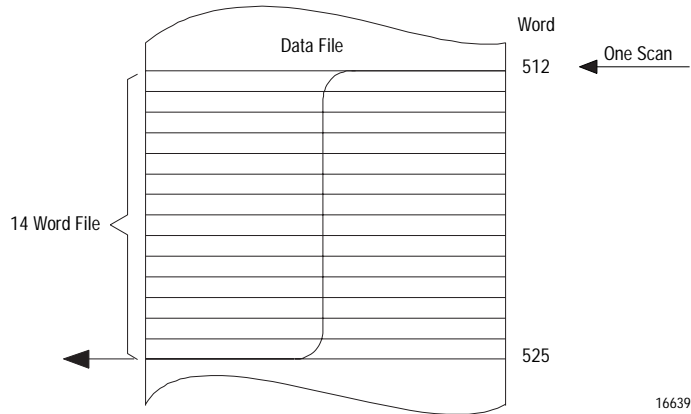
16617a

Choosing Modes of Block Operation

The block mode tells the processor how to distribute the block operation over one or more program scans. Select one of the following modes:

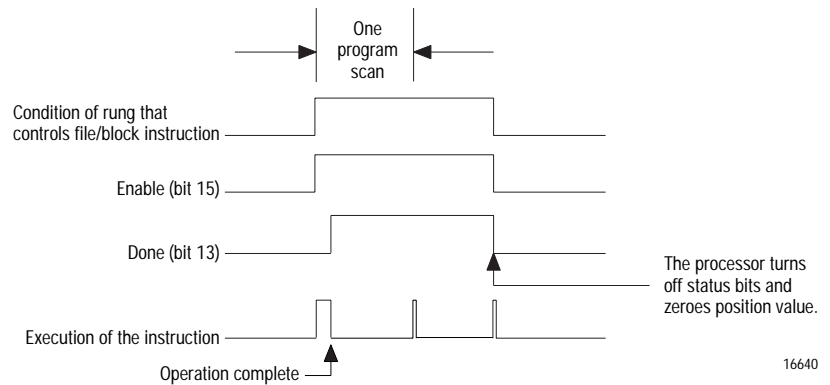
All Mode

In the **All** mode, the entire file is operated on before continuing on to the next rung of the program. Type an **A** for the mode parameter when you enter the instruction.



Operation begins when the rung goes from not true to true. The position (.POS) value in the control structure points to the word in the data block that the instruction is currently using. Operation stops when the function completes or when the processor detects an error.

The following timing diagram shows the relationship between status bits and instruction operation. When the instruction execution is complete, the done bit is turned on. The done and enable bits are not turned off, and the position value is not zeroed until the rung conditions are no longer true. Only then can another operation be triggered by a not-true-to-true transition of the rung conditions.

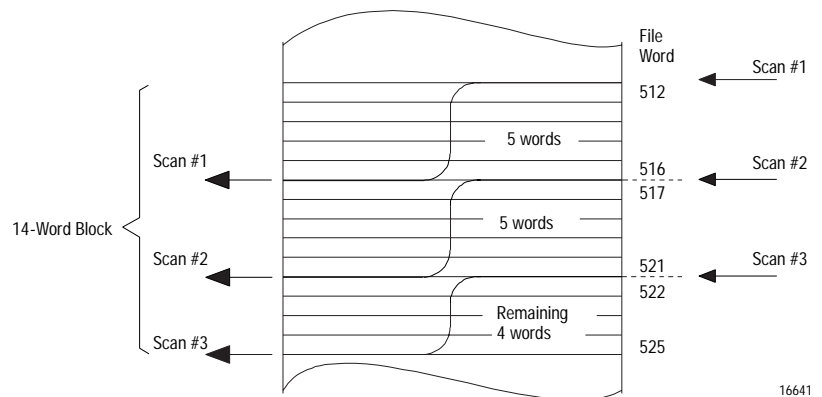


Numerical Mode

Numerical mode distributes the file operation over a number of program scans. To select the numerical mode, enter the **number of words per scan** (1-1000) for the mode parameter when you enter the file instruction. The number of words you enter must be less than or equal to the file length.

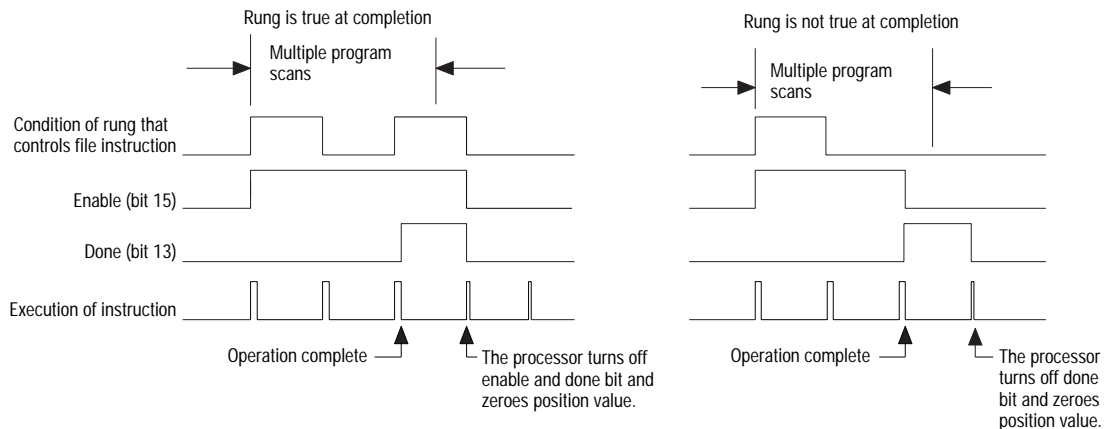
Execution is triggered when the rung conditions go from not true to true. Once triggered, the instruction is executed continually each time the rung is scanned in the program for the number of scans necessary to complete operation on the entire file. Once triggered, rung logic can change repeatedly without interrupting execution of the instruction.

Each time the rung is scanned, the instruction operates on the number of words equal to the rate you entered for the mode value, until it has operated on the number of words you specified by the length value. In the last scan of the rung, the processor may operate on less than the number of words you entered.



Important: Avoid using the results of a file instruction operating in numeric mode until the done bit is set because the data will be incomplete.

The following timing diagram shows the relationship between status bits and instruction operation.



When the instruction execution is complete, the done bit is turned on.

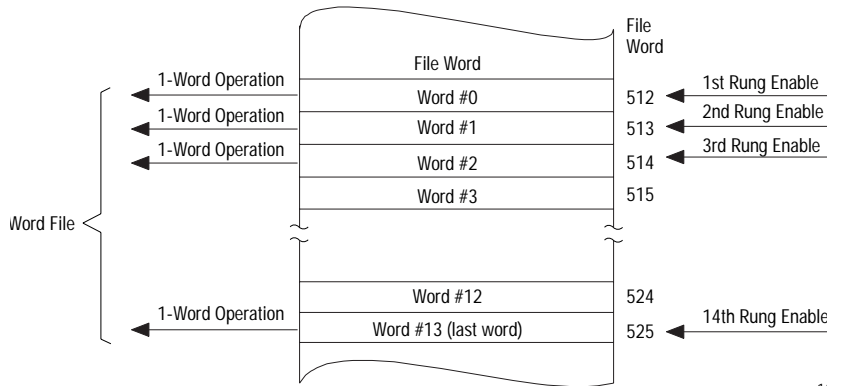
If the rung is true at completion, the enable bit and done bit are not turned off until the rung is no longer true. When the rung is no longer true, these bits are turned off and the position value is zeroed.

If the rung is not true at completion, the enable bit is turned off immediately, and one scan after the enable bit is turned off, the done bit is turned off and the position value is zeroed.

Only after the enable and done bits are turned off can another operation be triggered by a not-true-to-true transition of the rung conditions.

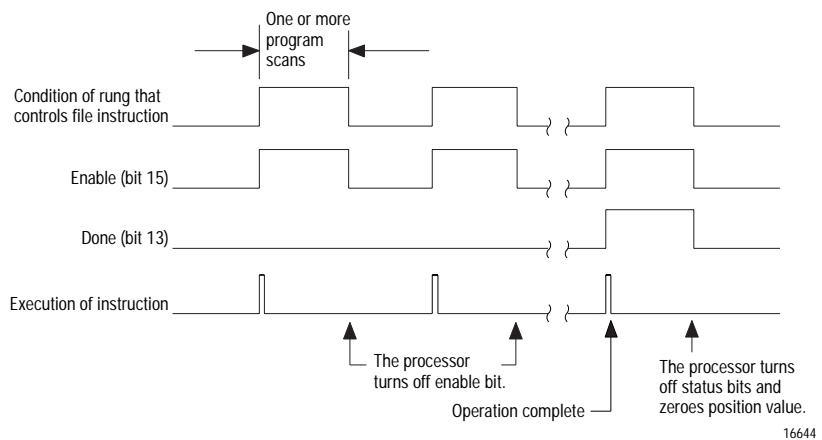
Incremental Mode

Incremental mode manipulates one word of the file each time the rung goes from not true to true. Type an **I** for the mode parameter when you enter the instruction.



14

The following timing diagram shows the relationship between status bits and instruction operation.



16644

Execution occurs only in a program scan in which the rung goes from not true to true. Each time this does occur, only one word of the file is operated on. The enable bit is on when rung logic is true. The done bit is turned on when the last word in the file has been operated on. When the last word in the file has been operated on and the rung goes from true to not true, the enable and done bits are turned off and the position value is zeroed. If the rung remains true for more than one program scan, the file instruction is not executed in subsequent scans after the transition.

Important: If you are operating on an entire file, avoid using the results of a file/block instruction using incremental mode until the done bit is on (the data will be incomplete).

Special Case, Numerical Mode with Words Per Scan = 1

The difference between numerical mode with a rate of 1 word per scan and incremental mode is:

- Numerical mode with any number of words per scan, only one not-true-to-true rung transition is required for continual execution of the instruction until operation is complete on the entire file.
- Incremental mode requires a not-true-to-true rung transition for each word in the file.

File Instructions FAL, FSC, COP, FLL

Using File Instructions

The file instructions perform operations on file data and compare file data. Table 9.A lists the available file instructions.

Table 9.A
Available File Instructions

If You Want to:	Use this Operation:	Found on Page:
Perform arithmetic, logic, shift, and function operations on file data	FAL	9-2
Perform search and compare operations on file data	FSC	9-14
Copy the contents of a file into another file	COP	9-19
Fill a file with specific values	FLL	9-20

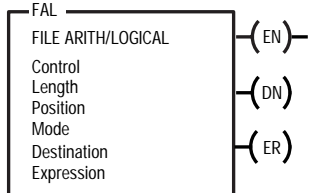
If you have not already done so, review the basic concepts of file operation in the previous chapter. For more information on using indexed addresses, see your software user manual.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

File Arithmetic and Logic (FAL)

The FAL instruction performs copy, arithmetic, logic, and function operations on the data stored in files. The FAL instruction performs the same operations as the CPT instruction. The difference is that the FAL instruction performs operations on multiple words, while the CPT instruction handles single words.

Description:



The FAL instruction is an output instruction that performs the operations defined by source addresses and operators you write in the expression. The instruction writes the results into a destination address.

Select how the processor distributes the operation over one or more program scans by your selection of instruction mode. For more information about modes of file operation, see chapter 8.

The FAL instruction automatically converts the data type at the source addresses to the data type that you specify in the destination address.

You can use this instruction to perform operations such as:

- zero a file
- copy data from one file to another
- make arithmetic or logic computations on data stored in files
- unload a file of error codes one at a time for display



ATTENTION: Instructions with a # sign in an address manipulate the offset value stored at S:24. Make sure you monitor or load the offset value you want prior to using an indexed address. Otherwise unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

Table 9.B
FAL Operations

Type	Operator	Description	Example Operation
Copy	none	copy from A to B	enter source address in the expression enter destination address in destination
Clear	none	set a value to zero	0 (enter 0 for the expression)
Arithmetic	+	add	2 + 3 2 + 3 + 7 (Enhanced PLC-5 processors)
	-	subtract	12 - 5 (12 - 5) - 1 (Enhanced PLC-5 processors)
	*	multiply	5 * 2 6 * (5 * 2) (Enhanced PLC-5 processors)
		divide	24 6 (24 6) * 2 (Enhanced PLC-5 processors)
	-	negate	- N7:0
	SQR	square root	SQR N7:0
	**	exponential (x to the power of y)	10**3 (Enhanced PLC-5 processors only)
Bitwise	AND	bitwise AND	D9:3 AND D10:4
	OR	bitwise OR	D9:4 OR D9:5
	XOR	bitwise exclusive OR	D10:10 XOR D10:11
	NOT	bitwise complement	NOT D9:4
Conversion	FRD	convert from BCD to binary	FRD D14:0
	TOD	convert from binary to BCD	TOD N7:0

Using Status Bits

To use the FAL instruction correctly, examine and control status bits in the control element. You address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	<p>by a false-to-true rung transition and indicates the instruction is enabled.</p> <p>In incremental mode, the .EN bit follows the rung condition. In numerical and ALL modes, the .EN bit remains set until the instruction completes its operation, regardless of the rung condition. The .EN bit is reset when the rung goes false and the instruction completes its operation.</p>
Done .DN (bit 13)	<p>after the instruction has operated on the last set of words.</p> <p>In numerical mode if the instruction is false at completion, it resets the .DN bit one program scan after the operation is complete. If the instruction is true at completion, the .DN bit is reset when the instruction goes false.</p>
Error .ER (bit 11)	<p>when the operation generates an overflow. The instruction stops until the ladder program resets the .ER bit.</p> <p>When the processor detects an error, the position value stores the number of the word that faulted.</p>

With the FAL instruction, a maximum of 80 characters of the expression can be displayed. If the expression you enter is near this 80 character maximum, when you accept the rung containing the instruction, the processor may expand it beyond 80 characters. When you try to edit the expression, only the first 80 characters are displayed and the rung is displayed as an error rung. The processor does contain the complete expression, however, and the instruction runs properly.

To work around this display problem, export the processor memory file and make your edits in the PC5 text file. Then import this text file. See your programming manual for more information on importing/exporting processor memory files.

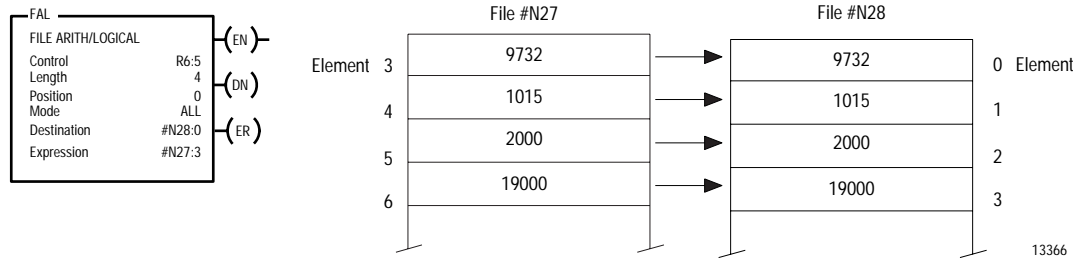
FAL Copy Operations

The FAL copy operation copies data:

- between files
- from a word to a file
- from a file to a word

To copy data with the FAL copy operation, enter the source address or program constant in the expression and the destination address in the destination.

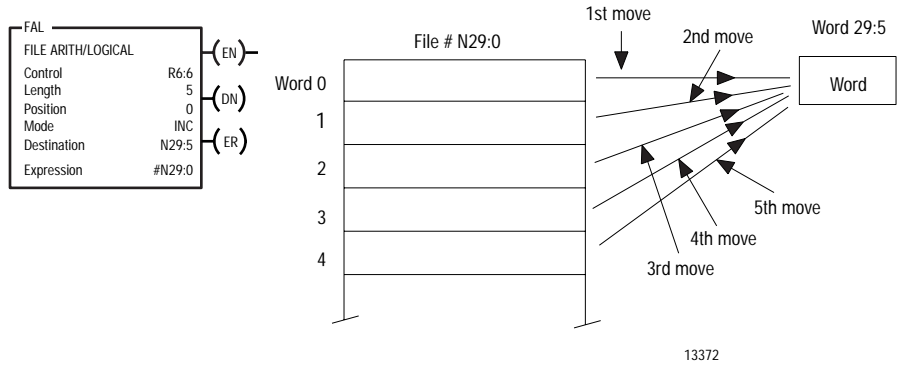
File-to-File Copy Example:



This Parameter:	Tells the Processor:
Control (R6:5)	What control structure controls the operation. This parameter is controlled by the rung condition, the state of the .EN and .DN bits, and by the mode (incremental, numeric, or all). It contains the location of the last value written to by the FAL instruction. For example, if, in incremental mode, position = 0 and length = 4, the last word written to by the FAL instruction would be word 3 since the instruction starts at location 0.
Length (4)	To move four words
Position (0)	To start at the source address
Mode (ALL)	To execute the length in one program scan
Destination (#N28:0)	Where to write the data (the # indicates that the operation is to be performed on a file)
Expression (#N27:3)	Where to read the data (the # indicates that the operation is to be performed on a file)

When the rung goes true, the processor reads four elements of integer file N27 word by word starting at element 3, and writes the image to integer file N28 starting at element 0. It writes over any data in the destination file.

File-to-Word Copy Example:



This Parameter:	Tells the Processor:
Control (R6:6)	What control structure controls the operation
Length (5)	To copy five words
Position (0)	To start at the source address
Mode (incremental)	To copy one word each time the rung goes true
Destination (N29:5)	Where to write the data (word address)
Expression (#N29:0)	Where to read the data (the # indicates that the operation is to be performed on a file)

With each false-to-true rung transition, the processor reads one element of integer file N29 starting at element 0, and writes the image into element 5 of integer file N29. The instruction writes over any data in the destination.

A word-to-file move is similar except that the instruction copies data from a word address into a file. The word address can be in the same or a different file.

FAL Arithmetic Operations

You can perform multiple arithmetic operations on file data (integer or floating point) with the following operators:

Operator:	Meaning:	Operator:	Meaning:
+	add		divide
-	subtract	-	negate
*	multiply	0	clear

For more information about order of operation, see chapter 4.

Upper and Lower Limits

The limits of data being mathematically manipulated depend on the type of file in which the data is stored. The following guidelines apply:

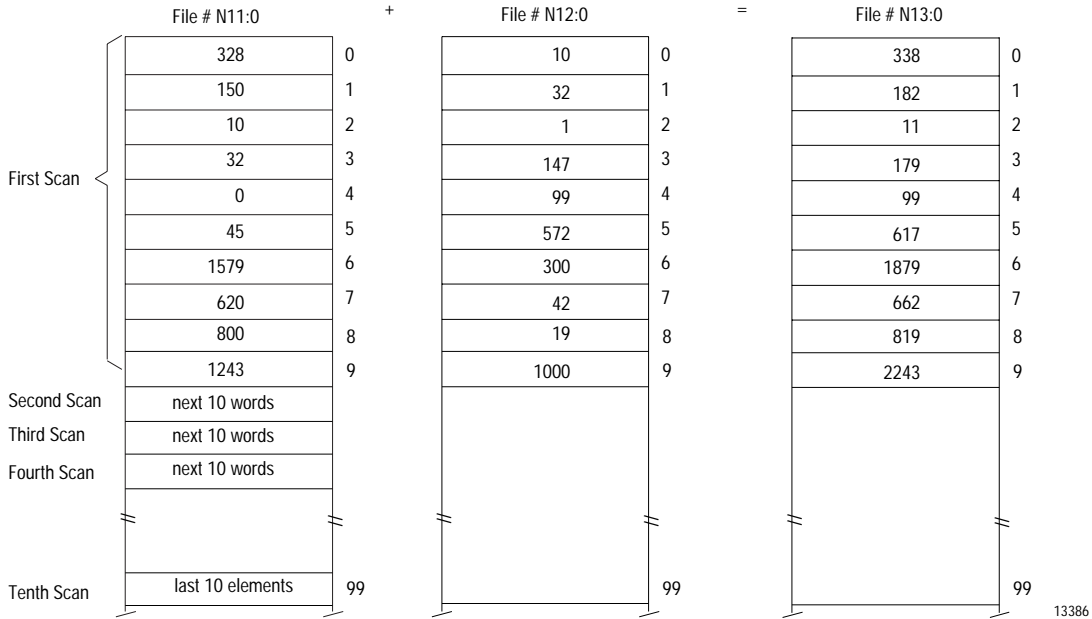
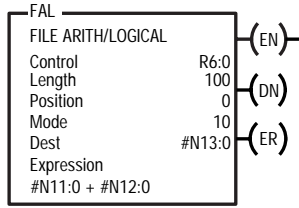
- all data except floating point is signed integer
- negative values are stored in two's complement
- floating point numbers are formatted as a subset of IEEE single-precision floating point

Type of File:	Range Stored in Word:
bit	-32,768 to +32,767 for integers
integer	-32,768 to +32,767
timer	0 to +32,767
counter	-32,768 to +32,767
control	0 to +32,767
floating point	$\pm 1.1754944e^{-38}$ to $\pm 3.4028237e^{+38}$

An error occurs when the result of an operation exceeds either the lower or upper limit of the destination word in which it is stored. The overflow bit is set in the processor's status file (S:0/1). The instruction also sets the error bit in the status byte of its control word.

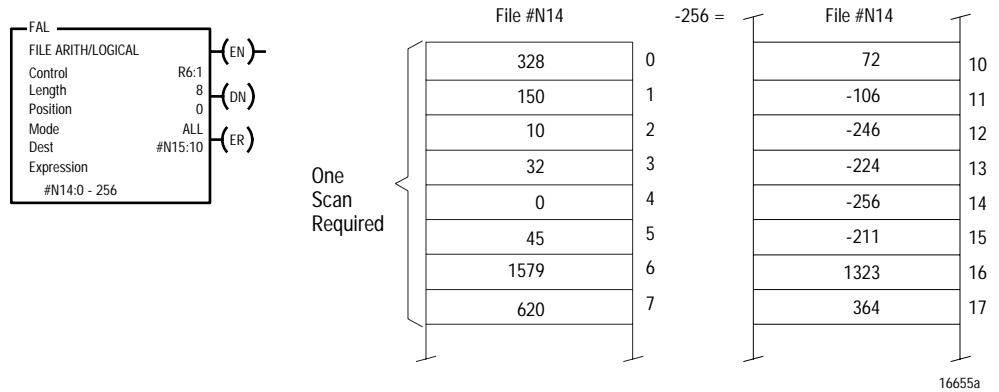
Addition Example:

When the rung goes true, the processor adds 100 values in file #N11:0 to the corresponding values in file #N12:0, using the numerical mode of 10 words per scan. The operation is performed in 10 scans and the instruction sequentially adds the values in the expression, storing the result in file #N13:0.



This Parameter:	Tells the Processor:
Control (R6:0)	What control structure controls operation
Length (100)	To operate on one hundred elements
Position (0)	To start at the source address
Mode (10)	To execute the data in 10 words per scan
Destination (#N13:0)	Where to write the result data
Expression (#N11:0 + #N12:0)	The operators, program constants, and source addresses

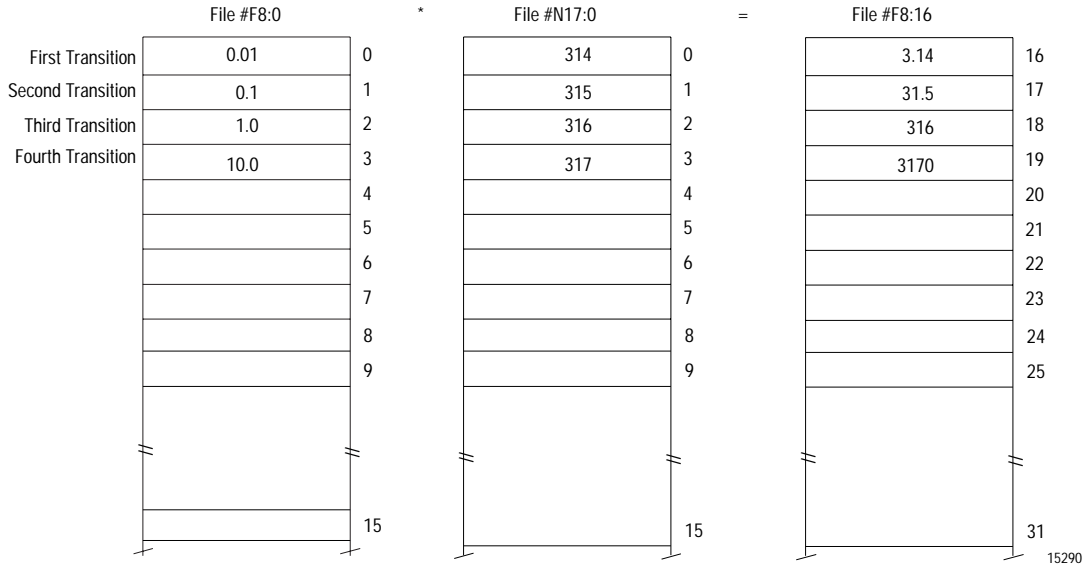
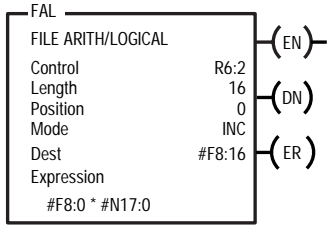
Subtraction Example:



This Parameter:	Tells the Processor:
Control (R6:1)	What control structure controls operation
Length (8)	To operate on eight words
Position (0)	To start at the source address
Mode (ALL)	To execute the data in one program scan
Destination (#N15:10)	Where to write the result data
Expression (#N14:0 - 256)	The operators, program constants, and source addresses

When the rung goes true, the processor reads eight elements of integer file N14 word by word starting at element 0, subtracts a program constant (256) from each, and writes the result into destination file N15 starting at element 10, all in one scan.

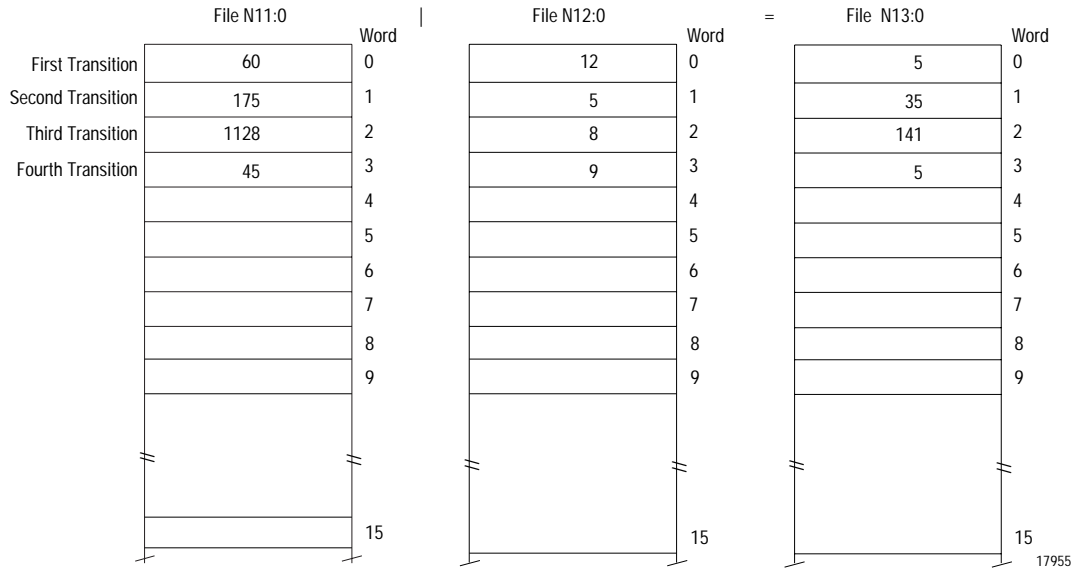
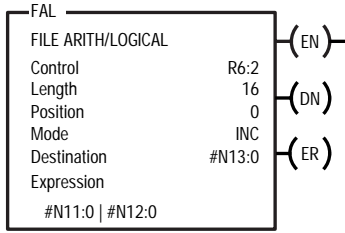
Multiplication Example:



This Parameter:	Tells the Processor:
Control (R6:2)	What control structure controls operation
Length (16)	To operate on sixteen words
Position (0)	To start at the source address
Mode (incremental)	To execute using incremental mode
Destination (#F8:16)	Where to write the result data
Expression (#F8:0 * #N17:0)	The operators, program constants, and source addresses

When the rung goes true, the processor multiplies 16 values in file #F8:0 by the corresponding values in file #N17:0, using incremental mode. One multiplication is performed for each false to true transition. The operation requires 16 transitions, storing the result in file #F8:16.

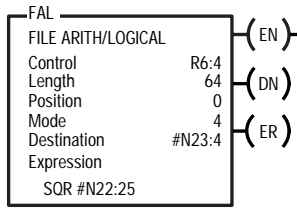
Division Example:



This Parameter:	Tells the Processor:
Control (R6:2)	What control structure controls operation
Length (16)	To operate on sixteen words
Position (0)	To start at the source address
Mode (incremental)	To execute using incremental mode
Destination (#N13:0)	Where to write the result data
Expression (#N11:0 #N12:0)	The operators and source addresses

When the rung goes true, the processor starts to divide 16 values starting at N11:0 by the corresponding values in file #N12:0, using incremental mode. One division is performed for each transition to true. The operation requires 16 transitions, storing the result in a 16-word file starting at N13:0.

File Square Root Example:



When rung conditions go true, the instruction obtains the positive square root of the value at the source. The rate is determined by the mode you select. The result of each square root operation is stored in the corresponding word in the destination, one word at a time.

The processor takes the square root of the absolute value (if the sign is negative, the processor disregards the sign).

This Parameter:	Tells the Processor:
Control (R6:4)	What control structure controls the operation
Length (64)	To take the square root of 64 words
Position (0)	To start at the source address
Mode (4)	To operate on 4 words each scan
Destination (#N23:4)	Where to write the result data
Expression (SOR #N22:25)	The operator and source address

After rung goes true, the square root of the first 4 words in the file beginning at N22:25 is calculated, and the result is written in the destination file beginning at N23:4. Every time the rung is scanned thereafter, the next four words are calculated and the result written to the destination file. The processor requires a total of 16 scans (length = 64 / mode = 4) to complete the instruction.

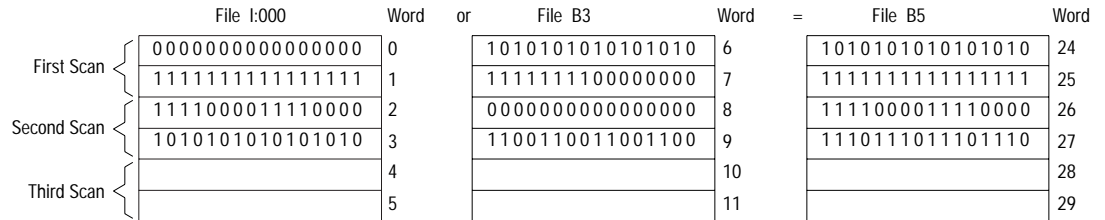
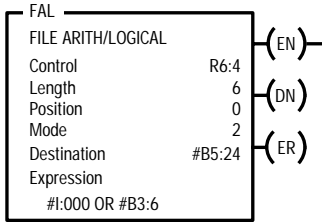
FAL Logic Operations

Perform multiple logic operations on binary file data with the following bitwise logic operators:

- AND
- OR
- XOR
- NOT

To perform multiple logic operations, you enter the operators, source addresses, or program constants in the expression, and the result address in the destination.

Logical OR Example:



16618a

This Parameter:	Tells the Processor:
Control (R6:4)	What control structure controls the operation
Length (6)	To OR 6 words
Position (0)	To start at the source address
Mode (2)	To move 2 words each scan
Destination (#B5:24)	Where to write the result data
Expression (#I:000 OR #B3:6)	The operator(s) and source addresses

After rung goes true, the processor performs a logical or operation on two words beginning at I:0 and B3:6. The result is written in the destination file beginning at B5:24. Every time the rung is scanned thereafter, the next two words are calculated and the result written to the destination file. The processor requires a total of 3 scans (length = 6 / mode = 2) to complete the instruction.

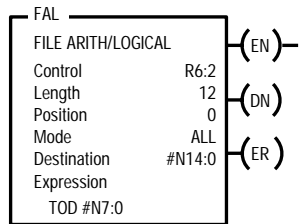
The processor executes logic operators in a predetermined order. For more information about order of operations, see chapter 4.

FAL Convert Operations

The FAL instruction can perform these convert operations:

- convert from integer to BCD (TOD)
- convert from BCD to integer (FRD)

Example: Convert to BCD



When rung conditions go to true, the processor converts the value in the source from integer to BCD. The rate is determined by the mode that you select. The result of the operation is stored in the corresponding word in the destination.

Example: Convert from BCD

When rung conditions go to true, the processor converts the value in the source from BCD to integer. The rate is determined by the mode that you select. The result of the operation is stored in the corresponding word in the destination.

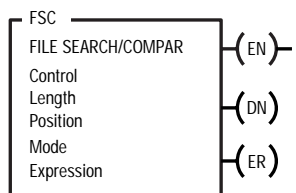
Important: Convert BCD values to integer before manipulating them; if you do not convert the values, the processor manipulates them as integer and their BCD value is lost.

File Search and Compare (FSC)

The FSC instruction performs search and compare operations.

These are the same operations as the CMP instruction, including complex expressions (Enhanced PLC-5 processors only). The difference is that the FSC instruction performs logical operations on files, while the CMP instruction operates on a single word. Also, the FSC instruction is an output instruction, while the CMP instruction is an input instruction.

Description:



The FSC instruction is an output instruction that compares values in source files, word by word, for the logical operations you specify in the expression. When the processor finds the specified comparison is true, it sets the found bit .FD, and records the position .POS where the true comparison was found. The inhibit bit .IN is set to prevent any further searching of the files.

Your ladder program must examine the found bit .FD and the position .POS to take appropriate action. Reset the inhibit bit .IN, so the instruction can continue.

Select how the processor distributes the operation over one or more program scans by your selection of instruction mode. For more information about modes of file operation, see chapter 8.

Use this instruction to perform operations such as:

- set high and low process alarms for multiple analog inputs
- compare batch variables against a reference file before starting a batch operation

Using Status Bits

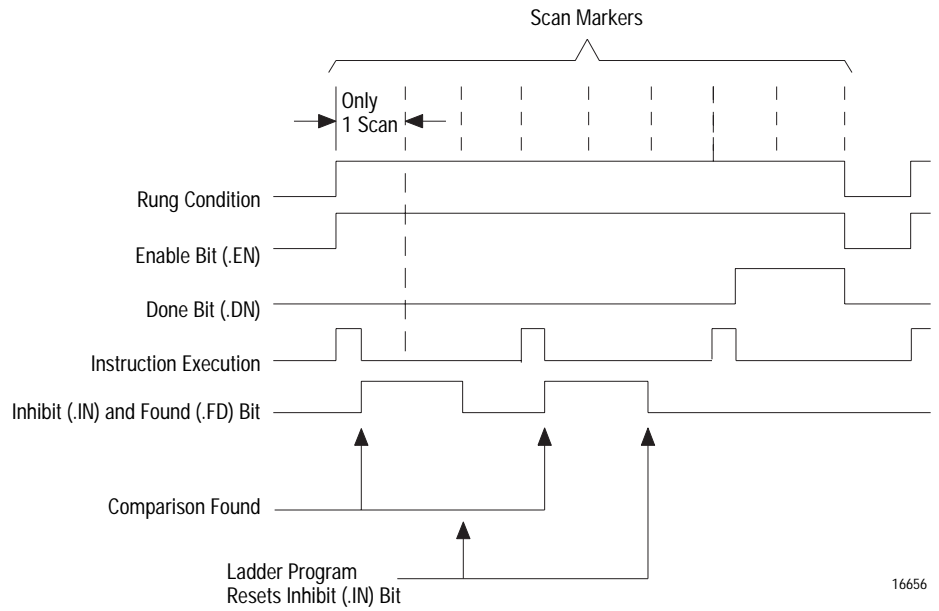
To use the FSC instruction correctly, your ladder program must examine and control status bits in the control structure. You must address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	by a false-to-true rung transition and indicates the instruction is enabled. In incremental mode this bit follows the rung condition. In Numerical and All modes, this bit remains set until the instruction completes its operation, regardless of the rung condition. The .EN bit is reset when rung conditions go false, but only after the instruction has set the .DN bit.
Done .DN (bit 13)	after the instruction has operated on the last set of words. In numerical mode if the instruction is false at completion, it resets the .DN bit one program scan after the operation is complete. If the instruction is true at completion, the .DN bit is reset when the instruction goes false.
Error .ER (bit 11)	when the operation generates an overflow. The instruction stops until the ladder program resets this bit. When the processor detects an error, the position value stores the number of the element that faulted.
Inhibit .IN (bit 9)	when the processor detects a true comparison. Your ladder program must reset this bit to continue the search after taking an action initiated by examining the .FD bit. The ladder program must reset this bit to continue operation.
Found .FD (bit 8)	when the processor detects a true comparison. The processor stops the search and also sets the inhibit .IN bit. The .FD bit is the output of the FSC instruction.

With the FSC instruction, a maximum of 80 characters of the expression can be displayed. If the expression you enter is near this 80 character maximum, when you accept the rung containing the instruction, the processor may expand it beyond 80 characters. When you try to edit the expression, only the first 80 characters are displayed and the rung is displayed as an error rung. The processor does contain the complete expression, however, and the instruction runs properly.

To work around this display problem, export the processor memory file and make your edits in the PC5 text file. Then import this text file. See your programming manual for more information on importing/exporting processor memory files.

The following timing diagram for All mode shows relationships between status bits and instruction execution when the instruction finds two true conditions.



16656

For more information about how the FSC instruction responds when it finds no true comparisons, see the timing diagrams in chapter 8.

FSC Search and Compare Operations

The FSC instruction performs these comparisons on file data according to how you specify them in the Expression. (Complex expressions are valid in Enhanced PLC-5 processors only.)

Comparison:	Example Expression:
Search Equal	#N50:0 = #N51:0
Search Not Equal	#N52:0 <> N52:11
Search Less Than	#B3:100 < #N53:0
Search Less Than or Equal	#F60:0 <= F60:12
Search Greater Than	#N54:0 > 256
Search Greater Than or Equal	F60:10 >= #N61:0

Data Conversion

The processor compares files of different data types by internally converting data into its binary equivalent before performing the comparison. The processor treats the following data types as integer: timer, status, bit, counter, input, ASCII, control, output, BCD.

Important: When you compare floating point and integer values in the FSC instruction, limit the comparisons to “less than or equal” and “greater than or equal.”

Important: Use ASCII and BCD for display only, and not as values. Since the processor interprets them as integer, they may lose their meaning if you enter them as values.

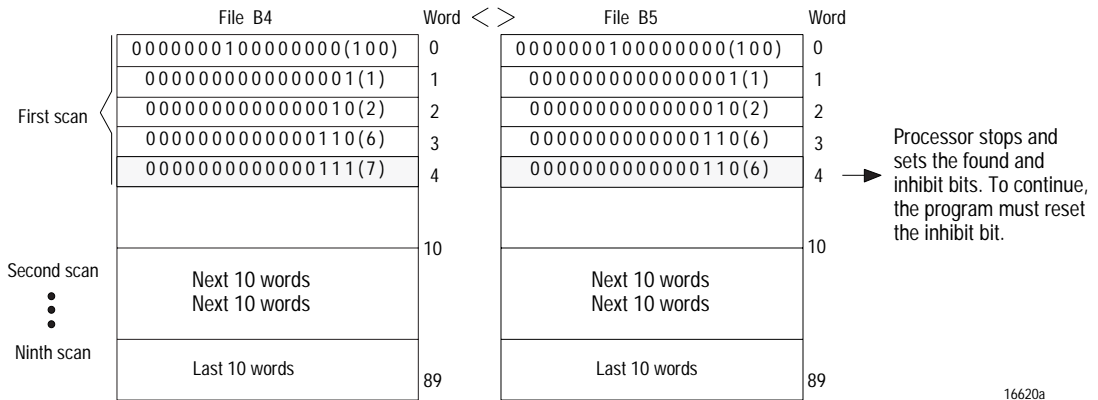
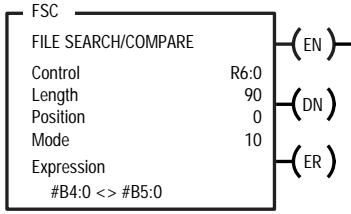
For the order in which the instruction performs logical operations, see the section “Determining the Order of Operation” in chapter 4.

File Search Operation

When the rung condition goes to true, the desired comparison is performed on data addressed in the expression. Words are compared in ascending order, starting at the beginning. The rate is determined by the mode of operation that you specify.

The .DN bit (bit 13) is set after the processor has compared the last pair. If the rung is true at completion, the .DN bit is turned off when the rung is no longer true. In numerical mode, however, if the rung is not true at completion, the .DN bit stays on one program scan after the operation is complete.

Example of Search Not Equal:



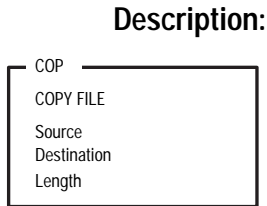
16620a

This Parameter:	Tells the Processor:
Control (R6:0)	What control structure controls the operation
Length (90)	To search through 90 words
Position (0)	To start at source addresses
Mode (10)	To search 10 words per program scan
Expression (#B4:0 <> #B5:0)	The comparison to perform and the source addresses

When a rung containing the FSC instruction goes to true, the processor performs the not-equal-to comparison between words, starting at B4:0 and B5:0. The number of words compared per program scan (10 in this example) is determined by the mode you select.

When the processor finds that corresponding source words are not equal (words B4:4 and B5:4 in this example), the processor stops the search and turns on the found .FD and inhibit .IN bits so your ladder program can take appropriate action. To continue the search comparison, you must turn off the .IN bit.

File Copy (COP)



The COP instruction is an output instruction that copies the values in the source file into the destination file. The source remains unchanged. The COP instruction does not use status bits. If you need an enable bit, program a parallel output that uses a storage address.

The COP instruction does not write over file boundaries. Any overflow data is lost. Also, no data conversion occurs if the source and destination files are different data types; use files of the same data type for each.

If the destination is in a file of words (such as an integer file) you specify the length in words. If the destination is in a file of structures (such as a counter file) you specify the length in structures. For example, if the source is in an integer file, the destination is in a counter file, and you specify a length of 5, 15 integer words are copied into 5 counter structures.

Entering Parameters

To program the COP instruction, you must provide the processor with the following:

Parameter:	Definition:
Source	the starting address of the source file. The source remains unchanged.
Destination	address of the destination file. The instruction writes over any data already stored at the destination.
Length	the number of the words/structures to overwrite in the destination file.

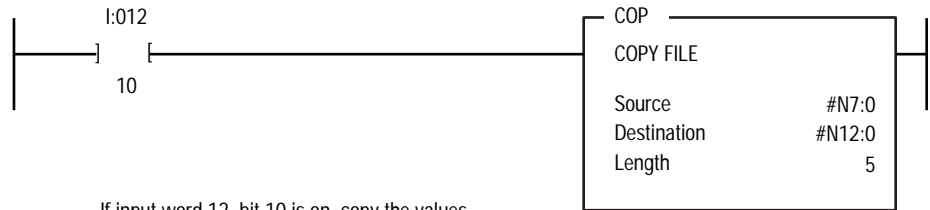


ATTENTION: If you use the COP instruction with an Enhanced PLC-5 processor, series A-D, file boundaries might become crossed if the destination parameter is indirectly addressed.

If the indirect address is written to the program area, the Enhanced PLC-5 processor, series A-D, displays major fault code 11 (bad user program checksum). If the indirect address is written outside of the program area, unexpected results could occur.

If you use the COP instruction with an Enhanced PLC-5 processors, series E and higher, this condition is correctly identified by either major fault code 20 (indirect address out of range high) or major fault code 21 (indirect address out of range low).

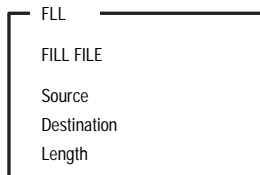
Example:



If input word 12, bit 10 is on, copy the values of the first five words starting at N7:0 into the first five words of N12:0.

File Fill (FLL)

Description:



The FLL instruction is an output instruction that fills the words of a file with a source value. The source remains unchanged. The FLL instruction does not use status bits. If you need an enable bit, program a parallel output that uses a storage address.

The FLL instruction does not write over file boundaries. Any overflow data is lost. Also, no data conversion occurs if the source and destination files are different data types; use files of the same data type for each.

If the destination is in a file of words (such as an integer file) you specify the length in words. If the destination is in a file of structures (such as a counter file) you specify the length in structures. For example, if the source word is an integer file, the destination is in a counter file, and you specify a length of 5, the source word is copied 15 times to fill the 5 counter structures.

The FLL instruction is level sensitive.

Entering Parameters

To program the FLL instruction, you must provide the processor with the following:

Parameter:	Definition:
Source	the address of the source word or a program constant. The source remains unchanged.
Destination	the starting address of the destination file. The instruction writes over any data already stored at the destination.
Length	the number of the words/structures to fill in the destination file

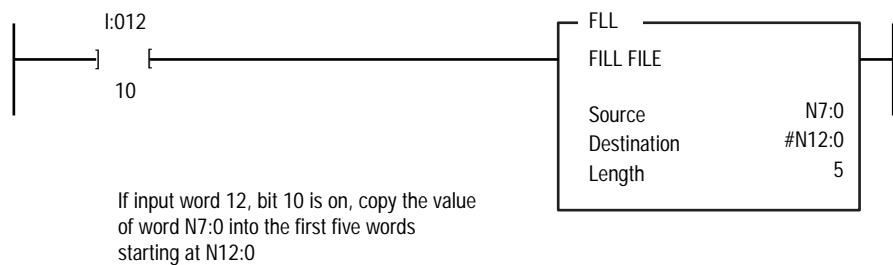


ATTENTION: If you use the FLL instruction with an Enhanced PLC-5 processor, series A-D, file boundaries might become crossed if the destination parameter is indirectly addressed.

If the indirect address is written to the program area, the Enhanced PLC-5 processor, series A-D, displays major fault code 11 (bad user program checksum). If the indirect address is written outside of the program area, unexpected results could occur.

If you use the FLL instruction with an Enhanced PLC-5 processors, series E and higher, this condition is correctly identified by either major fault code 20 (indirect address out of range high) or major fault code 21 (indirect address out of range low).

Example:



Words are copied from the specified source file into the specified destination file every scan that the rung is true. They are copied (in ascending order with no transformation of data) up to the specified number or until the last word of the destination file is reached, whichever occurs first.

Accurately specify the starting address and length of the data block you are filling. The instruction will not write over a file boundary (such as between files N16 and N17) at the destination. The overflow would be lost.

Notes:

Diagnostic Instructions FBC, DDT, DTR

Using Diagnostic Instructions

The diagnostic instructions let you detect problems with data in your programs. Table 10.A lists the available diagnostic instructions.

Table 10.A
Available Diagnostic Instructions

If You Want to:	Use this Operation:	Found on Page:
Compare I/O data against a known, good reference and record any mismatches	FBC	10-2
Compare I/O data against a known, good reference, record any mismatches, and update the reference file to match the source file	DDT	10-2
Pass source data through a mask and compare the result to reference data, and then write the source word into the reference address of the next comparison.	DTR	10-8

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

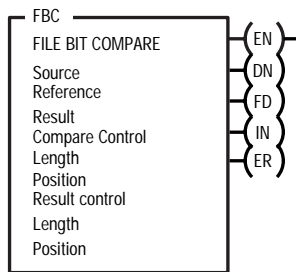
File Bit Comparison (FBC) and Diagnostic Detect (DDT)

The FBC and DDT diagnostic instructions are output instructions that you use to monitor machine or process operations to detect malfunctions.

Table 10.B
Available Diagnostic Instructions

If You Want to Detect Malfunctions By:	Use this Instruction:
Comparing bits in a file of real-time inputs with a reference bit file that represents correct operation	FBC
Change-of-state diagnostics	DDT

Description:



Both the FBC and DDT instructions compare bits in a file of real-time machine or process values (input file) with bits in a reference file, detect deviations, and record mismatched bit numbers. These instructions record the position of each mismatch found and place this information in the result file. If no mismatches are found, the .DN bit is set but the result file remains unchanged.

The difference between the DDT and FBC instruction is that each time the DDT instruction finds a mismatch, the processor changes the reference bit to match the source bit. The FBC instruction does not change the reference bit. Use the DDT instruction to update your reference file to reflect changing machine or process conditions.

Selecting the Search Mode

Select whether the diagnostic instruction searches for one mismatch at a time or whether it searches for all mismatches during one program scan.

One Mismatch at a Time

With each false-to-true rung transition, the instruction searches for the next mismatch between the input and reference files. Upon finding a mismatch, the instruction stops and sets the found .FD bit. Then the instruction enters the position number of the mismatch into the result file.

The DDT instruction also changes the status of the reference bit to match the status of the corresponding input bit. The instruction resets the found bit when the rung goes false.

When the instruction reaches the end of the file, the done bit (bit 13 DN of the compare control element) is set. Then, when the rung goes false, the instruction resets:

- enable bit
- found bit (if set)
- compare done bit
- result done bit (if set)
- both control counters

To enable this mode of operation, set the inhibit bit (.IN = 1) either by ladder program or manually before program execution.

All Per Scan

The instruction searches for all mismatches between the input and reference files in one program scan. Upon finding mismatches, the instruction enters the position numbers of mismatched bits into the result file in the order it finds them. After reaching the end of the input and reference files, the instruction sets the .FD bit if it finds at least one mismatch. Then the instruction sets the .DN bit.

If you use a result file that cannot hold all detected mismatches (if the result file fills), the instruction stops and requires another false-to-true rung transition to continue operation. The instruction wraps the new mismatched bit positions into the beginning of the result file writing over the old.

After completing the comparison and when the rung goes false, the instruction resets:

- enable bit
- found bit (if set)
- compare done bit
- result done bit (if set)
- both control counters

To enable this mode of operation, reset the inhibit bit (.IN = 0) by ladder program or manually before program execution.

Entering Parameters

To program these instructions, you need to provide the processor with the following information:

Parameter:	Description:
Source	the indexed address of your input file.
Reference	the indexed address of the file that contains the data with which you compare your input file.
Result	the indexed address of the file where the instruction stores the position (bit) number of each detected mismatch.
Cmp Control	the address of the comparison control structure (R) that stores status bits, the length of the source and reference files (both should be the same), and the current position during operation. Use the compare control address with mnemonic when you address these parameters: Length (.LEN) is the decimal number of bits to be compared in the source and reference files. Remember that bits in I/O files are numbered in octal 00-17, but that bits in all other files are numbered in decimal 0-15. Position (.POS) is the current position of the bit to which the instruction points. Enter a value only if you want the instruction to start at an offset concurrent with a control file offset for one scan.
Result Control	the address of the result control structure (R) that stores the bit position number each time the instruction finds a mismatch between source and reference files.

Use the result control address with mnemonic when you address these parameters:

- **Length (.LEN)** is the decimal number of elements in the result file. Make the length long enough to record the maximum number of expected mismatches.
- **Position (.POS)** is the current position in the result file. Enter a value only if you want the instruction to start at an offset concurrent with a control file offset for one scan.



ATTENTION: Do not use the same address for more than one control structure. Duplication of these addresses could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

Using Status Bits

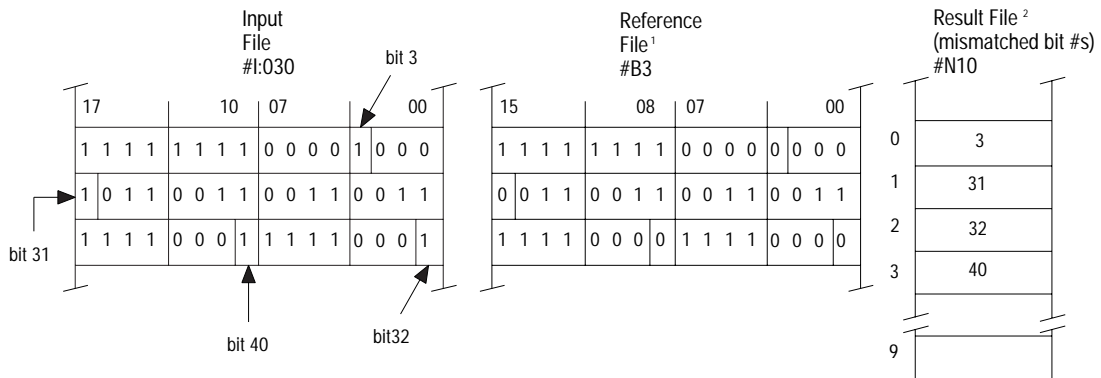
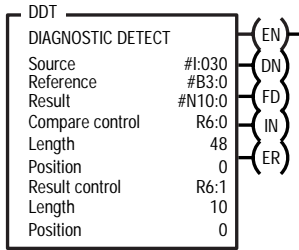
To use the FBC or DDT instruction correctly, examine and control bits in both the comparison and result control elements. You address these bits by mnemonic.

Bit:	Function:	
Comparison Control Bits	Enable .EN (bit 15)	starts operation on a false-to-true rung transition If the .IN bit is set for one-at-a-time operation, the ladder program must toggle the .EN bit after the instruction detects each mismatch.
	Done .DN (bit 13)	is set when the processor reaches the end of the source and reference files
	Error .ER (bit 11)	is set when the processor detects an error and stops operation of the instruction For example, an error occurs if the length (.LEN) is less than or equal to zero or if the position (.POS) is less than zero. The ladder program must reset the .ER bit if the instruction detects an error.
	Inhibit .IN (bit 09)	determines the mode of operation When this bit is reset, the processor detects all mismatches in one scan. When this bit is set, the processor stops the search at each mismatch and waits for the ladder program to re-enable the instruction before continuing the search.
Result Control Bits	Found .FD (bit 08)	is set each time the processor records a mismatch bit number in the result file (one-at-a-time operation) or after recording all mismatches (all per scan).
	Done .DN (bit 13)	is set when the result file fills The instruction stops and requires another false-to-true rung transition to reset the result .DN bit and then continue. If the instruction finds another mismatch, it wraps the new position number around to the beginning of the file, writing over previous position numbers.

After the FBC or DDT instruction sets the compare .DN bit, the instruction is reset when the rung's input conditions go false. The instruction resets its status bits and both control elements.

Example:

The DDT instruction above compares the bits in the source file (#I:030) with the bits in the reference file (#B3:0), recording the mismatched bit positions in the result file (#N10:0):



The FBC and DDT instructions detect mismatches and record their locations by bit number in a result file.

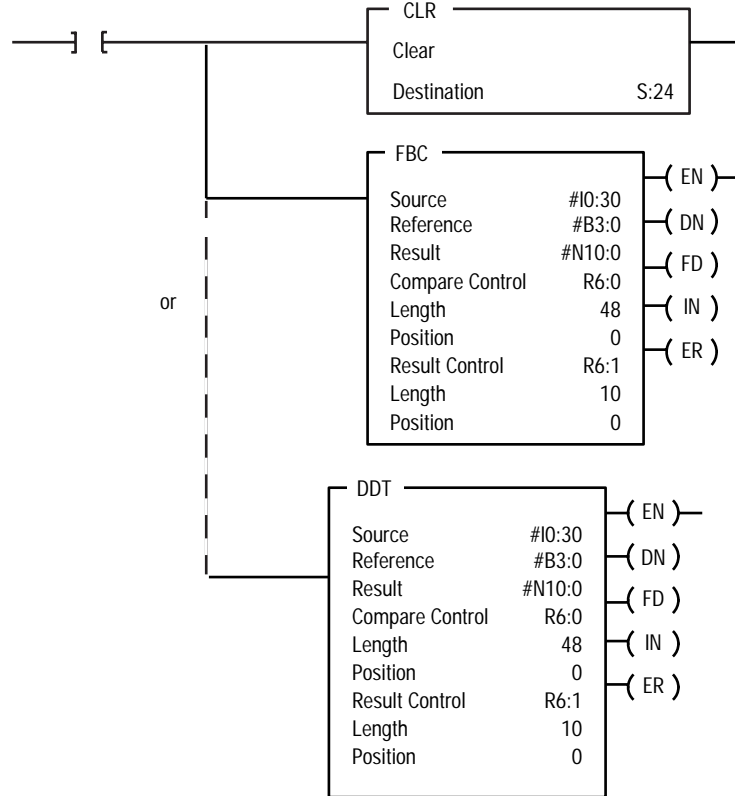
¹ The DDT instruction changes the status of the corresponding bit in the reference file to match the input file when it detects a mismatch.

² The length of the result file is the length that you enter for RESULT CONTROL.

16657a

This Parameter:	Tells the Processor:
Source (#I:030)	Where to find input data for comparison
Reference (#B3:0)	Where to find the reference file
Result (#N10:0)	Where to store mismatched bit numbers
CMP Control (R6:0)	What control structure controls the comparison
Length (48)	The number of bits to be compared
Position (0)	To start at the beginning of the file
Result Control (R6:1)	What control structure controls the result
Length (10)	The number of words reserved for mismatches
Position (0)	To start at the beginning of the file

Important: The FBC and DDT instructions may cause any Enhanced PLC-5 processor to fault if the indexed addressing offset contains a value that exceeds data table boundaries. To work around this, add a ladder rung that clears S:24 (indexed addressing offset) immediately before an FBC or DDT instruction.



Data Transitional (DTR)

Description:

DTR
DATA TRANSITION
Source
Mask
Reference

The DTR instruction is an input instruction that passes a source value through a mask and compares the result to a reference value. Use this instruction to detect and identify invalid inputs and to prevent invalid inputs from shutting down a batch processor or machine operation.

The DTR instruction compares a source word through a mask with a reference word. The instruction also writes the source word into the reference address for the next comparison. The source word remains unchanged.

When the masked source differs from the reference, the instruction goes true for only one scan. The processor writes the masked source value into the reference address. When the masked source and the reference are the same, the instruction remains false.



ATTENTION: Online programming with this instruction can be dangerous. If the destination value is different from the source value, the instruction goes true. Use caution if you insert this instruction when the processor is in Run or Remote Run mode.

Entering Parameters

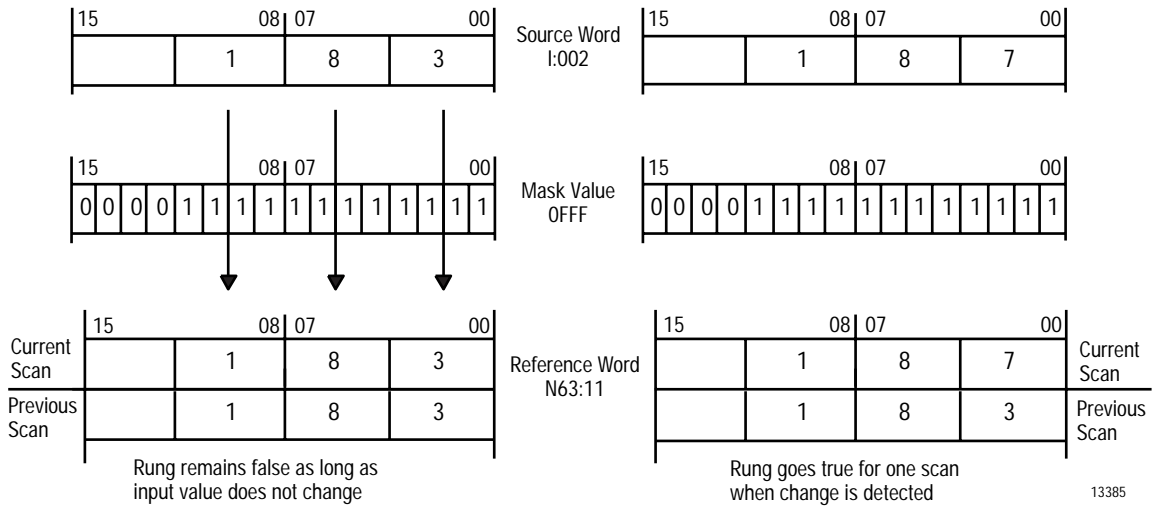
To program the DTR instruction, you need to provide the processor with the following information:

Parameter:	Definition:
Source	the address of the input word, typically real inputs.
Mask	the hexadecimal value or address that contains the mask value
Reference	the address of the reference word The reference contains the source data from the last DTR scan

Example:

DTR	
DATA TRANSITION	
Source	I:002
Mask	OFFF
Reference	N63:11

The DTR instruction above passes the source (I:002) through a mask of OFFF and compares the result to the reference word (N63:11). The source word is then written into the reference address for the next comparison (the source remains unchanged).



Notes:

Shift Register Instructions BSL, BSR, FFL, FFU, LFL, LFU

Applying Shift Registers

Use the shift register instruction to simulate the movement or flow of parts and information.

If You Use a Shift Register for:	Data in the Shift Register Could Represent:
Tracking parts through an assembly line	Part types, quality, size, and status
Controlling machine or process operations	The order in which events occur
Inventory control	Identification numbers or locations
System diagnostics	A fault condition that caused a shutdown

Table 11.A lists the available shift instructions.

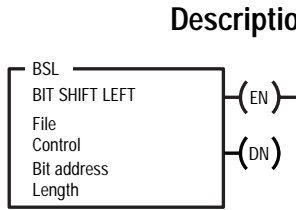
Table 11.A
Available Shift Instructions

If You Want to:	Use these Instructions:	Found on Page:
Load bits into, shift bits through, and unload bits from a bit array one bit at a time, such as for tracking bottles through a bottling line where each bit represents a bottle	BSL, BSR	11-2
Load and unload values in the same order, such as for tracking parts through an assembly line where parts are represented by values that have a part number and assembly code	FFL, FFU	11-5
Load and unload values in reverse order, such as tracking stacked inventory in a warehouse, where goods are represented by serial number and inventory codes	LFL, LFU *	11-8

* These instructions are only supported by Enhanced PLC-5 processors.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Bit Shift Instructions



Description:

Bit shift instructions shift all bits within the specified address one bit position with each false-to-true rung transition. These instructions are:

- Bit Shift Left (BSL)
- Bit Shift Right (BSR)

Entering Parameters

To program a bit shift instruction, you need to provide the processor with the following information:

Parameter:	Definition:
File	the address of the bit array you want to manipulate. You must start the array at a 16-bit word boundary. For example, use bit 0 of word number 1, 2, 3, etc. You can end the array at any bit number up to 15,999. However, you cannot use the remaining bits in that particular element because the instruction invalidates them.
Control	The address of the control structure (48 bits – three 16-bit words) in the control area (R) of memory that stores the instruction's status bits, the size of the array (number of bits), and the bit pointer.
Position	the current position of the bit to which the instruction points. Enter a value only if you want the instruction to start at an offset concurrent with a control file offset for one scan. Use the control address with mnemonic when you address this parameter.
Bit Address	the address of the source bit. The instruction inserts the status of this bit in either the first (lowest) bit position (for the BSL instruction) or the last (highest) bit position (for the BSR instruction) in the array.
Length	the decimal number of bits to be shifted. Remember that bits in I/O files are numbered in octal 00-17, but that bits in all other files are numbered in decimal 0-15. Use the control address with mnemonic when you address this parameter.



ATTENTION: Do not use the same control address for more than one instruction. Unexpected operation could result in possible equipment damage and/or personal injury.

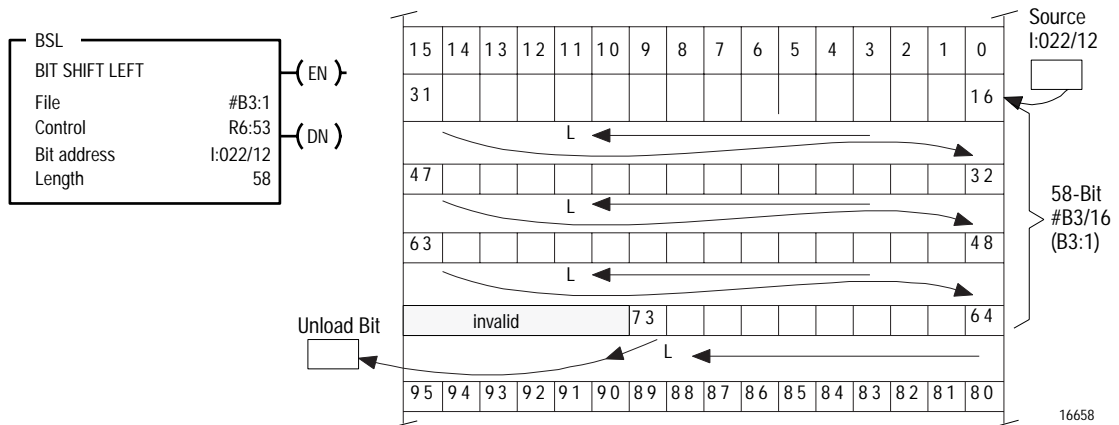
Using Status Bits

To use the BSL or BSR instruction correctly, examine status bits in the control element. You address these bits by mnemonic.

Bit:	Definition:
Enable .EN (bit 15)	is set when the rung makes a false-to-true rung transition to indicate the instruction is enabled.
Done .DN (bit 13)	is set to indicate that the bit array shifted one bit position
Error .ER (bit 11)	is set to indicate that the instruction detected an error, such as if you entered a negative file length
Unload .UL (bit 10)	is the instruction's output. The .UL bit stores the status of the bit removed from the array each time the instruction is enabled. Avoid using the .UL bit when the .ER bit is set.

Important: When enabled, the bit pointer is set to the value of the length the bit array is shifted. After all of the bits are shifted, the instruction resets the .EN, .ER and .DN bits and the bit pointer when input conditions go false.

Bit Shift Left (BSL) Example:



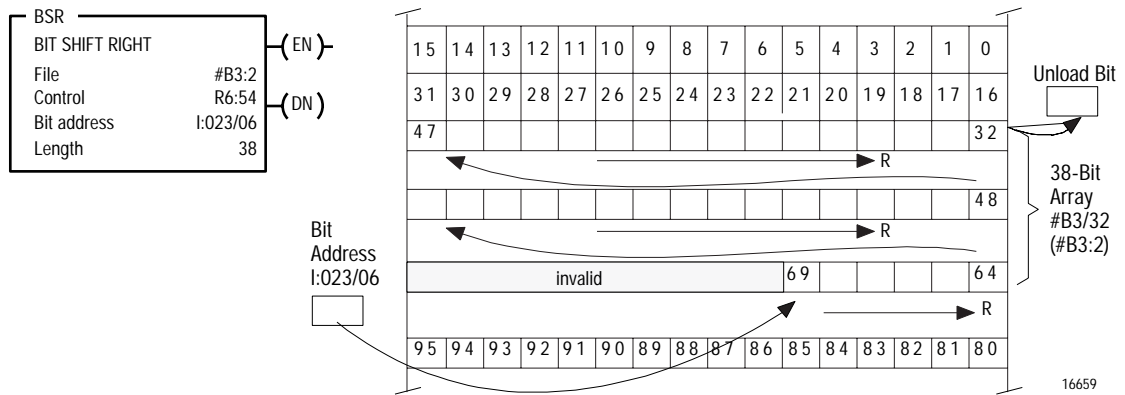
This Parameter:	Tells the Processor:
File (#B3:1)	The location of the bit array
Control (R6:53)	The instruction's address and control element
Bit Address (I:022/12)	The location of the source bit (bit 12 of input word 22)
Length (58)	The number of bits in the bit array

When a rung containing the BSL instruction goes from false to true, the processor sets the .EN bit. Then the processor shifts 58 bits in bit file B3, starting with bit 16, to the left (higher bit number) one bit position. The last bit shifts out at bit position 73 into the .UL bit. The specified source bit, bit 12 of input word 22, shifts into the first bit position, bit 16 of bit file B3.

After the processor completes the shift operation in one program scan, when the rung goes false, the instruction resets the .EN, .ER (if set) and .DN bits, and resets the pointer.

For wrap-around operation, make the source address the same as the highest (outgoing) bit address. You can omit using the .UL bit in wrap-around operation.

Bit Shift Right (BSR) Example:



This Parameter:	Tells the Processor:
File (#B3:2)	The location of the bit array
Control (R6:54)	The instruction's address and control element
Bit Address (I:023/06)	The source bit address (bit 06 in input word 23)
Length (38)	The number of bits in the bit array

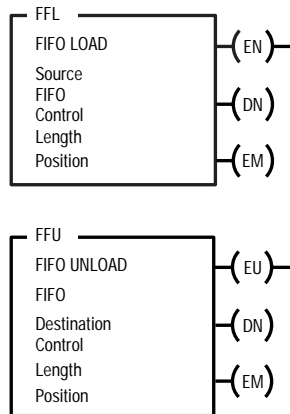
When a rung containing the BSR instruction goes from false to true, the processor sets the .EN bit. Then the processor shifts 38 bits in bit file B3 to the right (to a lower bit number) one bit position starting with the highest bit position 69. The lowest bit (bit 32) shifts out of the bit array into the .UL bit. The specified source, bit 06 of input word 23, shifts into the highest bit position 69.

After the processor completes the shift operation in one program scan, when the rung goes false, the instruction resets the .EN, .ER (if set) and .DN bits, and resets the pointer.

For wrap-around operation, make the source address the same as the lowest (outgoing) bit address. You can omit using the .UL bit in wrap-around operation.

Using FIFO and LIFO Instructions

Description:



Use FIFO instructions, First In – First Out (FFL and FFU) and LIFO instructions, Last In – First Out (LFL and LFU) in pairs to store and retrieve data in a prescribed order.

These Instructions:	Retrieve Data:
FFL and FFU	In the order stored (first in, first out)
LFL and LFU *	In reverse of the order stored (last in, first out)

* Available in Enhanced PLC-5 processors only

When used in pairs, these instructions establish an asynchronous shift register (stack).

Entering Parameters

When you program a FIFO or LIFO stack, use the same file and control addresses, length, and position values for both instructions in the pair. You need to provide the processor with the following information:

- **Source** is the address that stores the “next in” value to the stack. The FIFO or LIFO load instruction (FFL or LFL) retrieves the value from this address and loads it into the next word in the stack.
- **Destination** is the address that stores the value that exits from the stack.

This Instruction:	Unloads the Value from:
FIFO's FFU	Word zero
LIFO's LFU	The last word entered

- **FIFO** or **LIFO** is an indexed address of the stack. Use the same FIFO address for the associated FFL and FFU instructions; use the same LIFO address for the associated LFL and LFU instructions.
- **Control** is the address of the control structure (48 bits – three 16-bit words) in the control area (R) of memory. The control structure stores the instruction's status bits, stack length, and next available position (pointer) in the stack.

Use the control address with mnemonic when you address the following parameters:

- **Length (.LEN)** is the maximum number of elements in the stack.
- **Position (.POS)** indicates the next available location where the instruction loads data into the stack.

- **Length** specifies the maximum number of words in the stack. Address the length value by mnemonic .LEN.
- **Position** indicates the next available location where the instruction loads data into the stack. Address the position value by mnemonic .POS.

Enter a position value only if you want the instruction to start at an offset at power-up. Otherwise, enter 0. Your ladder program can change the position if necessary.



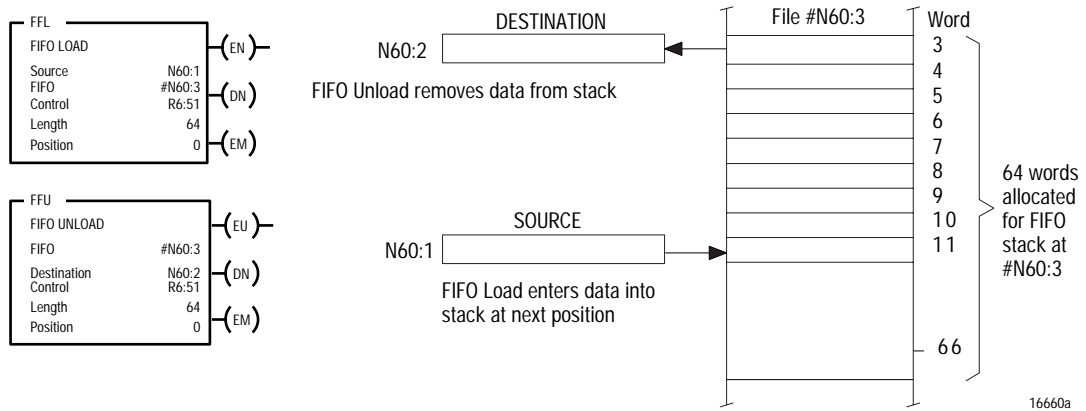
ATTENTION: Except when pairing stack instructions, do not use the same control address for any other instruction. Unexpected operation could result with possible equipment damage and/or personal injury.

Using Status Bits

To use the FIFO and LIFO instructions correctly, examine status bits in the control structure. You address these bits by mnemonic.

This Bit:	Is Set:
Enable Load .EN (bit 15)	when the rung makes a false-to-true rung transition to indicate the instruction is enabled (used in FFL and LFL instructions). Note: During prescan, this bit is set to prevent a false load when the program scan begins.
Enable Unload .EU (bit 14)	when rung conditions are true to indicate the instruction is enabled (used in FFU and LFU instructions). Note: During prescan, this bit is set to prevent a false unload when the program scan begins.
Done .DN (bit 13)	by the processor to indicate that the stack is full. The .DN bit inhibits loading the stack until there is room.
Empty .EM (bit 12)	by the processor to indicate that the stack is empty. Do not enable the FIFO or LIFO unload commands if the .EM bit is set.

FIFO Load (FFL) and FIFO Unload (FFU) Example:



This Parameter:	Tells the Processor:
Source (N60:1)	The location of the "next in" source word
FIFO (#N60:3)	The location of the stack (FIFO file)
Destination (N60:2)	The location of the "exit" word
Control (R6:51)	The instruction's address and control structure
Length (64)	The maximum number of words you can load
Position (0)	To start at the FIFO file address

FIFO Load Description:

When the rung that contains the FFL instruction goes from false to true, the processor sets the .EN bit and loads the source element (N60:1) into the next available element in the stack as pointed to by the control structure's position. The processor loads an element each time the rung goes from false to true, until it fills the stack. When the stack becomes full, the processor sets the .DN bit. The ladder program should detect that the stack is full and inhibit further loading of data from the source.

You may want to load the stack in advance, or enable the load instruction while inhibiting the unload instruction until the stack contains the desired data.

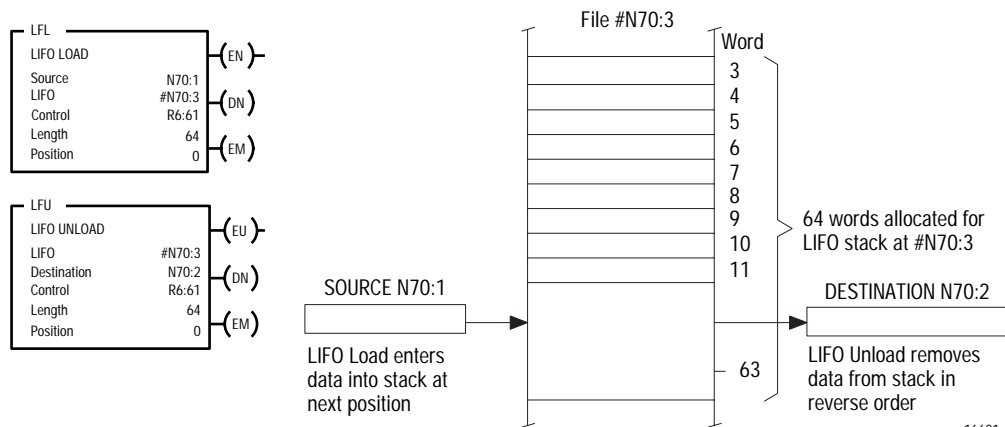
FIFO Unload Description:

When the rung that contains the FFU instruction goes from false to true, the processor sets the .EU bit and unloads data from the first element stored in the FIFO stack into the destination word N60:2. At the same time the processor shifts all data in the stack one position toward the first word. The processor unloads one word each time the rung goes from false to true until it empties the FIFO stack.

When the stack becomes empty, the processor sets the .EM bit. Thereafter, the processor transfers a zero value for each false-to-true rung transition until the FFL instruction loads new values. Your ladder program should detect that the stack is empty and inhibit other instructions from using zero values stored at the destination.

With a FFU instruction, you can unload data from a word other than the first word of the stack by changing the FIFO address to the address of the desired word and changing the length accordingly.

LIFO Load (LFL) and LIFO Unload (LFU)
Example:
(Enhanced PLC-5 processors only)



16621

This Parameter:	Tells the Processor:
Source (N70:1)	the location of the "next in" source word
LIFO (#N70:3)	the location of the stack (LIFO file)
Destination (N70:2)	the location of the "exit" word
Control (R6:61)	the instruction's control structure
Length (64)	the maximum number of words you can load
Position (0)	to start at the LIFO file address

Important: The difference between FIFO and LIFO stack operation is that the LFU instruction removes data in the reverse order to the order it is loaded (last-in-first-out). Otherwise, LIFO instructions operate identical to FIFO instructions.

LIFO Load Description: When the rung that contains the LFL instruction goes from false to true, the processor sets the .EN bit and loads the source word (N70:1) into the next available word in the stack as pointed to by the control structure's position. The processor loads an element each time the rung goes from false to true until it fills the stack. When the stack becomes full, the processor sets the .DN bit. The ladder program should detect that the stack is full and inhibit further loading of data from the source.

You may want to load the stack in advance or enable the load instruction while inhibiting the unload instruction until the stack contains the desired data.

LIFO Unload Description: When the rung that contains the LFU instruction goes from false to true, the processor sets the .EU bit and unloads data starting with the last word stored in the LIFO stack into the destination word N70:2. The processor unloads one word each time the rung goes from false to true until it empties the LIFO stack.

When the stack becomes empty, the processor sets the .EM bit. Thereafter, the processor transfers a zero value for each false-to-true rung transition until the load instruction loads new values. Your ladder program should detect that the stack is empty and inhibit other instructions from using zero values stored at the destination.

With a LIFO unload instruction, you can unload data from a word other than the first word of the stack by changing the LIFO address to the address of the desired word and changing the length accordingly.

Notes:

Sequencer Instructions SQO, SQI, SQL

Applying Sequencers

Sequencer instructions are typically used to control automatic assembly machines that have a consistent and repeatable operation. Use the sequencer input instruction to detect when a step is complete; use the sequencer output instruction to set output conditions for the next step. Use the sequencer load instruction to load reference conditions into the sequencer input and output file.

Table 12.A lists the available sequencer instructions.

Table 12.A
Available Sequencer Instructions

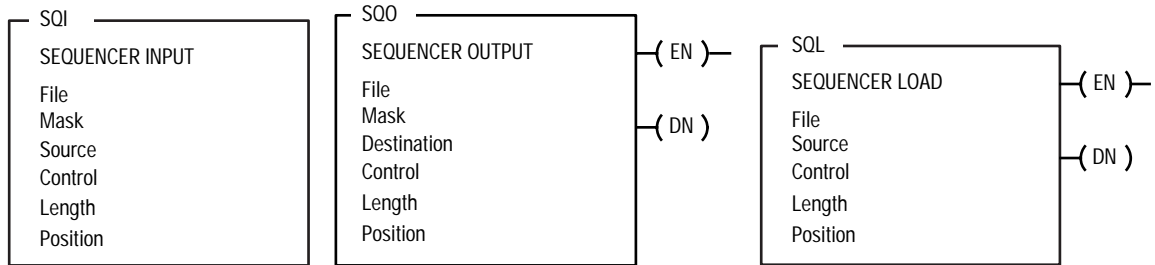
If You Want to:	Use this Instruction:	Found on Page:
Control sequential machine operations by transferring 16-bit data through a mask to output image addresses	SQO	12-5
Monitor machine operating conditions for diagnostic purposes by comparing 16-bit image data (through a mask) with data in a reference file	SQI	12-7
Capture reference conditions by manually stepping the machine through its operating sequences and loading I/O or storage data into destination files	SQL	12-8

Sequencer instructions can conserve program memory. These instructions monitor and control multiples of 16 discrete outputs at a time in a single rung.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Sequencer Instructions

Description: Use the SQI and SQO instructions in pairs to respectively monitor and control a sequential operation. Use the SQL instruction to load data in the sequencer file.



These instructions operate on multiples of 16 bits at a time. Place SQI instructions in series and SQO instructions in parallel in the same rung for 32-, 48-, 64-, or other bit operations.

Important: Each SQO instruction increments the control structure, so corresponding SQI instructions may miss parts of the source file.

Entering Parameters

When programming SQI and SQO instructions in pairs, use the same control address, length value, and position value in each instruction. The same applies when using multiple instructions in the same rung to double, triple, or further increase the number of bits.

To program sequencer instructions, you need to provide the processor with the following information:

- **File** is the indexed address of the sequencer file to or from which the instruction transfers data. Its purpose depends on the instruction:

In this Instruction:	The Sequencer File Stores Data for:
SQO	Controlling outputs
SQI	Reference to detect completion of a step or a fault condition
SQL	Creating the SQO or SQI file

- **Mask** (for SQO and SQI) is a hexadecimal code or the address of the mask element or file through which the instruction moves data. Set (1) mask bits to pass data; reset (0) mask bits to prevent the instruction from operating on corresponding destination bits. Specify a hexadecimal value for a constant mask value. Store the mask in an element or file if you want to change the mask according to application requirements.
- **Source** (for SQI and SQL) is the address of the input element or file from which the instruction obtains data for its sequencer file.
- **Destination** (for SQO, only) is the destination address of the output word or file to which the instruction moves data from its sequencer file.

Important: If you use a file for the source, mask, or destination of a sequencer instruction, the instruction automatically determines the file length and moves through the file step-by-step as it moves through the sequencer file.

- **Control** is the address of the control structure in the control area (R) of memory (48 bits – three 16-bit words) that stores the instruction's status bits, the length of the sequencer file, and the instantaneous position in the file.

Use the control address with mnemonic when you address the following parameters:

- **Length (.LEN)** is the length of the sequencer file.
- **Position (.POS)** is the current position of the word in the sequencer file that the processor is using.

For this Instruction:	The Control Structure Is Incremented:
SQO and SQL	By the instruction itself
SQI	Externally, either by the paired SQO with the same control address, or by another instruction



ATTENTION: Except for paired instructions, do not use the same control address for any other purpose. Duplication of a control element could result in unpredictable operation, possibly causing equipment damage and/or injury to personnel.

- **Length** is the number of steps of the sequencer file starting at position 1. Position 0 is the start-up position. The instruction resets to position 1 at each completion.

Important: The address assigned for a sequencer file is step zero. Sequencer instructions use (length + 1) words of data for each file referenced in the instruction. This also applies to the source, mask, and destination values if addressed as files.

- **Position** is the word location in the sequencer file. The position value is incremented internally by SQO and SQL instructions.

Important: Your ladder program can externally increment the position value of the SQI instruction. One way to do this is to pair it with the SQO instruction and assign the same control structure to both instructions.

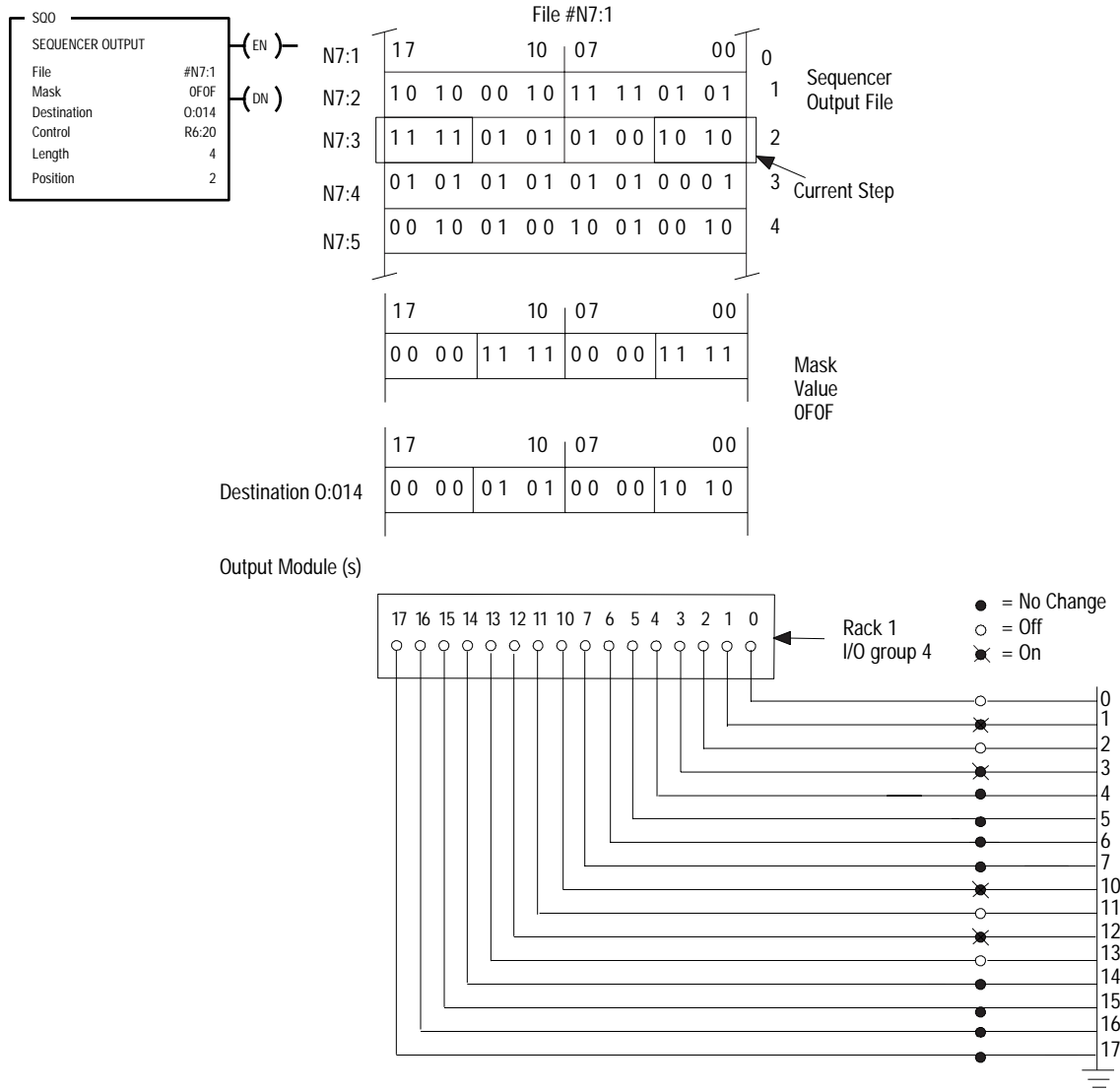
In earlier series processors, if the .POS value was out of range, the .POS value was automatically set to 1, which is the first step in the sequence. There was no indication that this occurred. In series E and later processors, if the .POS value exceeds the number of words in the file, the .ER bit is set, no data is written, and the .POS value remains the same.

Using Status Bits

To use the sequencer instructions correctly, the ladder program must examine status bits in the control element. You address these bits by mnemonic.

This Bit:	Is Set:
Enable .EN (bit 15)	(SQO or SQL) is set on a false-to-true rung transition to indicate that the instruction is enabled. The instruction follows the rung condition. Note: During prescan, this bit is set to prevent a false increment of the table pointer when the program scan begins.
Done .DN (bit 13)	(SQO or SQL) is set after the instruction finishes operating on the last word in the sequencer file. After the rung goes false, the processor resets the .DN bit on the next false-to-true rung transition.
Error .ER (bit 11)	when the length value is less than or equal to zero or when the position value is less than zero

Sequencer Output (SQO) Example:



SQO instruction moves the data of the current step through a mask to an output word for controlling multiple outputs.

This Parameter:	Tells the Processor:
File (#N7:1)	The location of the sequencer file
Mask (0F0F)	The fixed hexadecimal value of the mask
Destination (O:014)	The output image address to be changed
Control (R6:20)	The structure that controls the operation
Length (4)	The number of words to step through
Position (2)	The current position

The SQO instruction steps through the sequencer file of 16-bit output words whose bits have been set to control various output devices.

When the rung goes from false to true, the instruction increments to the next step (word) in the sequencer file #N7:1. The data in the sequencer file is transferred through a fixed mask (0F0F) to the destination address O:014. Current data is written to the destination element every scan that the rung remains true.

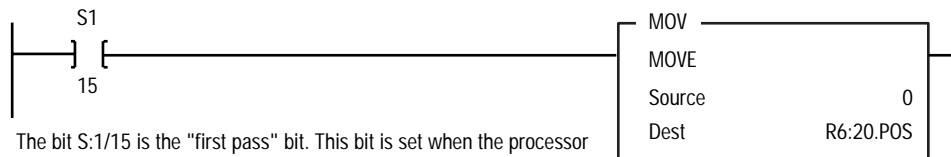
At start-up when you switch the processor from Program mode to Run mode, instruction operation depends on whether the rung is true or false on the first scan:

- If the rung is true and POS = 0, the instruction transfers data in step 0.
- If rung is false, the instruction waits for the first false-to-true rung transition and transfers data in step 1.

After transferring the last word of the sequencer file, the processor sets the .DN bit. On the next false-to-true rung transition, the processor resets the .DN bit and sets the position to step 1.

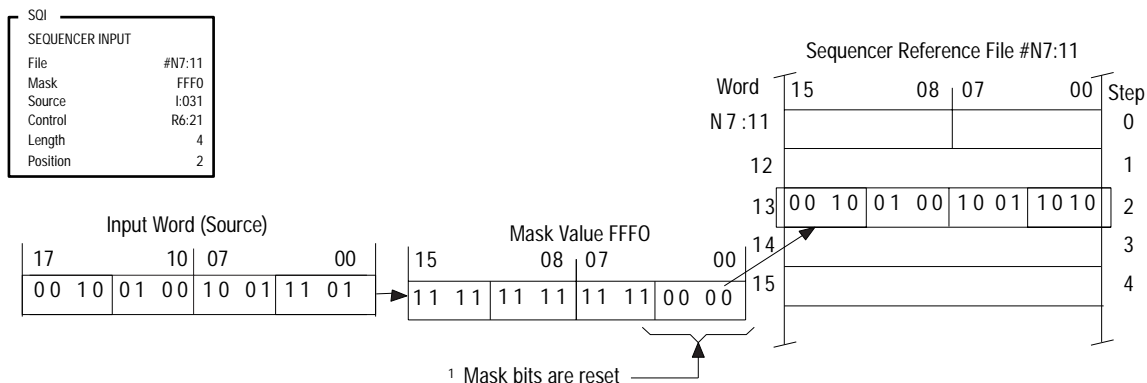
Resetting the Position of SQO

Each time the processor goes from Program to Run mode, you should reset the position of any SQO instruction. To do this, use the following ladder logic:



The bit S:1/15 is the "first pass" bit. This bit is set when the processor first scans a program. When this rung goes true, the processor moves the value of 0 to the position word of the SQO instruction. After the position is set to 0, the next false to true transition will cause the processor to run step 1.

Sequencer Input (SQI) Example:



SQI instruction is true when it detects that an input word matches (through a mask) its corresponding reference word.

¹ These bits are not compared. Therefore, the instruction is true in this example.

16646a

This Parameter:	Tells the Processor:
File (#N7:11)	The location of the reference file
Mask (FFF0)	The fixed hexadecimal value of the mask
Source (#I:031)	The input image address to be compared
Control (R6:21)	The element that controls the operation
Length (4)	The number of elements to step through
Position (2)	The current position

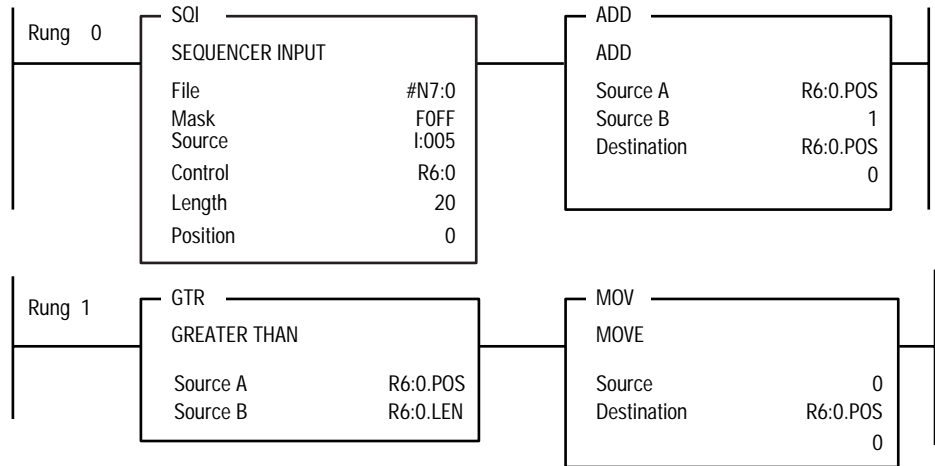
The SQI instruction compares a file of input image data (I:031), through a mask (FFF0), to a file of reference data (N7:11) for equality. When the status of all non-masked bits of the word at that particular step match those of the corresponding reference word, the instruction goes true. Otherwise, the instruction is false.

Important: You can use the SQI instruction with the control structure of the SQ0 instruction. Program the SQI as the condition instruction in the same rung with the SQ0. Assign the same control address and length to both instructions so they track together.

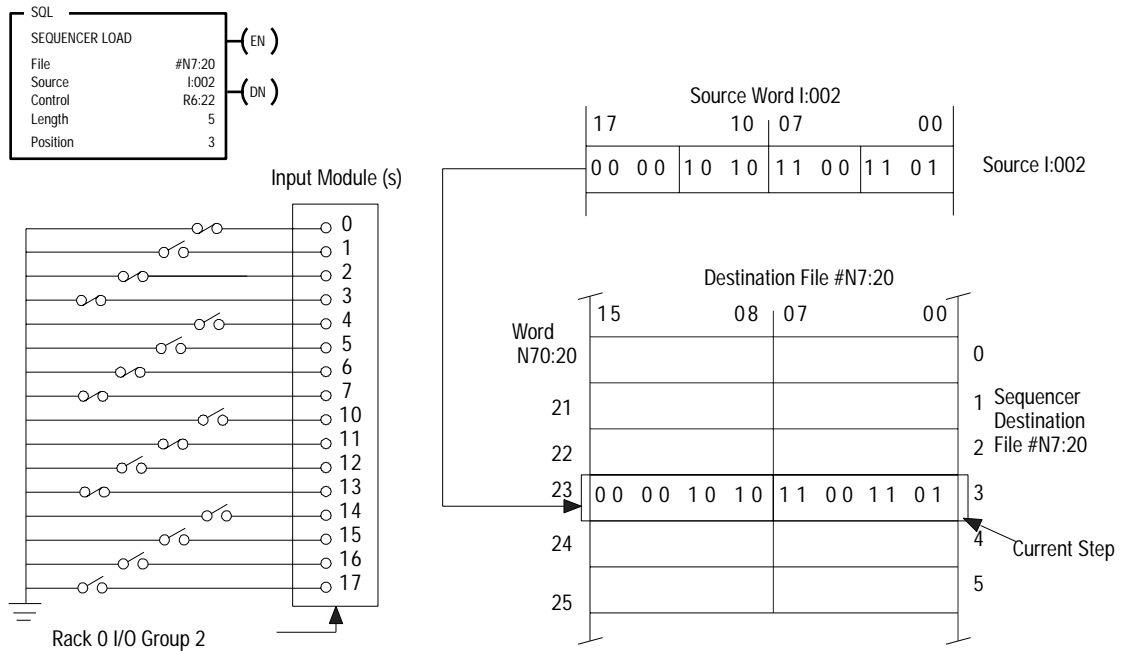
Using SQI Without SQ0

Another application of the SQI instruction is machine diagnostics where you load the reference file with data representing the desired sequence of machine operation. When operating, if the real time sequence of operation does not match the desired sequence of operation stored in the reference file, enable a fault signal. In this case, the ladder program externally increments the SQI instruction.

To externally increment the sequencer file, use a CPT instruction to move a new position value into the SQI instruction's control element. Do this to increment each step in the SQI instruction's file. Rung 0 increments the SQI instruction. Rung 1 resets the position value after stepping through the file.



Sequencer Load (SQL) Example:



SQL instruction loads data from the input word into a destination file from where it can be moved to other sequencer files.

16661a

This Parameter:	Tells the Processor:
File (#N7:20)	The location of the destination file
Source (I:002)	The input image address to be read
Control (R6:22)	The structure that controls the operation
Length (5)	The number of words to step through
Position (3)	The current step

When the rung goes from false to true, the SQL instruction increments to the next step in the sequencer file and loads data into it, one step for each rung transition. The SQL instruction loads current data each scan that the rung remains true. A mask is not used.

At start-up, when you switch the processor from Program to Run mode, the instruction operation depends on whether the rung is true or false on the first scan:

- If the rung is true, the instruction loads data into step 0.
- If the rung is false, the instruction waits for the first false-to-true rung transition and loads data into step 1.

After loading the last step, the processor sets the .DN bit. On the next false-to-true transition, the processor resets its .DN bit, resets the position to step 1, and loads data into that word.

Notes:

Program Control Instructions MCR, JMP, LBL, FOR, NXT, BRK, JSR, SBR, RET, TND, AFI, ONS, OSR, OSF, SFR, EOT, UIE, UID

Selecting Program Flow Instructions

Program flow instructions change the flow of ladder program execution. Use Table 13.A to select the program control instruction or group of instructions that fit your programming requirements.

Table 13.A
Available Program Control Instructions

If You Want to:	Then Use these Instructions:	Found on Page:
Turn off all non-retentive outputs in a section of ladder program	MCR	13-2
Jump over a section of a program that does not always need to be executed	JMP, LBL	13-3
Loop through a set of rungs for a preset number of times	FOR, NXT, BRK	13-5
Jump to a separate subroutine file, pass data to the subroutine, perform an operation, and return the results	JSR, SBR, RET	13-8
Mark a temporary end that halts program execution beyond it	TND	13-13
Disable a rung	AFI	13-13
Trigger a one-shot event based on a change in rung condition	ONS, OSR,* OSF*	13-14 (ONS), 13-15 (OSR), 13-16 (OSF)
Reset a sequential function chart	SFR*	13-17
End a transition file	EOT	13-18
Enable or disable user interrupts	UIE,* UID*	13-19 (UID), 13-20 (UIE)

*These instructions are only supported by Enhanced PLC-5 processors.

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Master Control Reset (MCR)

Description:



Use MCR instructions in pairs to create program zones that turn off all the non-retentive outputs in the zone. Rungs within the MCR zone are still scanned, but scan time is reduced due to the false state of non-retentive outputs. Non-retentive outputs are reset when their rung goes false.

**If the MCR Rung
that Starts the
Zone Is:**

Then the Processor:

True

Executes the rungs in the MCR zone based on each rung's individual input conditions (as if the zone did not exist).

False

Resets all non-retentive output instructions in the MCR zone regardless of each rung's individual input conditions.

MCR zones let you enable or inhibit segments of your program, such as for recipe applications.

When you program MCR instructions, 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 it activates the zone.
- If an MCR zone continues to the end of the ladder program, you do not have to program an MCR instruction to end the zone.

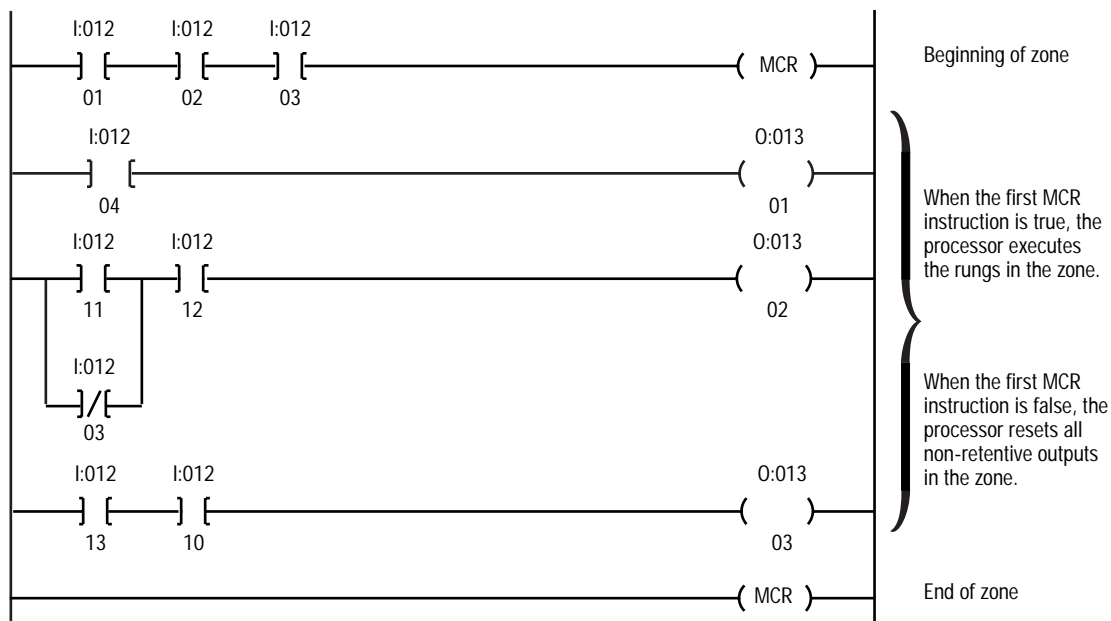
Important: The MCR instruction is not a substitute for a hard-wired master control relay that provides emergency stop capability. You still should install a hard-wired master control relay to provide emergency I/O power shutdown.



ATTENTION: Do not overlap or nest MCR zones. Each MCR zone must be separate and complete. If overlapped or nested, unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

ATTENTION: If you start instructions such as timers or counters in an MCR zone, instruction operation ceases when the zone is disabled. Re-program critical operations outside the zone if necessary.

Example: When the rung containing the first MCR instruction is true, the processor executes the rungs in the MCR zone based on the rung input conditions. Otherwise, the processor resets the non-retentive output instructions within the MCR zone.



Jump (JMP) and Label (LBL)

Description: Use JMP and LBL instructions in pairs to skip portions of the ladder program.



If the Jump Rung Is:	Then the Processor:
True	Skips from the JMP rung to the LBL rung and continues executing the program. You can jump forward or backward.
False	Ignores the JMP instruction

Jumping forward to a label saves program scan time by omitting a program segment until needed. Jumping backward lets the processor repeat iterations through a program segment until its logic is complete.

Important: Be careful not to jump backwards an excessive number of times. The watchdog timer could time out, which would fault the processor.

Using JMP

The JMP instruction lets the processor skip rungs. You can jump to the same label from one or more JMP instructions.



ATTENTION: Jumped timers and counters are not scanned. Re-program critical operations outside the jumped zone.

Using LBL

The LBL instruction is the target of the JMP instruction that has the same label number. Place the LBL instruction first on the rung to where you want the processor to jump.

Important: Make sure that the LBL instruction is the first instruction on the rung. (The software currently lets you to create a branch around an LBL instruction; this will cause the processor to operate incorrectly.)

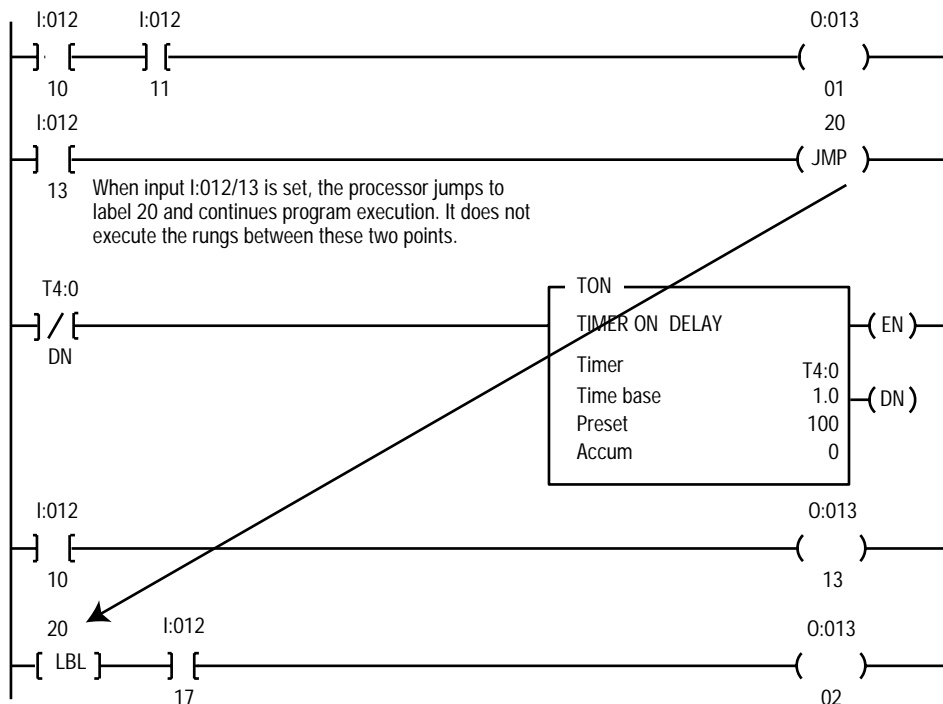
If You Have this Processor:	Valid LBL Numbers:	Valid Amount Per Program File:
Enhanced PLC-5	000-255	256
Classic PLC-5	0-31	32

If you modify and accept a rung containing a label while on line with the processor in Run mode, the software creates an I/R pair. If you modify the I rung before assembling the edits, the processor will fault with a duplicate label error.

There are four ways to avoid this problem:

- Edit the rung with the processor in Program mode.
- Cancel the edits and re-edit the rung.
- Let the fault occur, then clear the fault after assembling the edits.
- Assemble the first edit, then modify the rung again to make the second change. If you are editing on line, the processor may execute the rung with the first edit and may cause the processor to fault or run improperly.

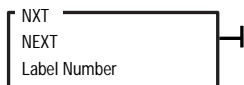
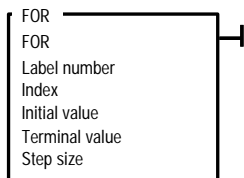
JMP and LBL Example: When the rung containing the JMP instruction goes true, the processor jumps over successive rungs until it reaches the rung that contains the LBL instruction with the same number. The processor resumes executing at the LBL rung.



The timer (TON) will not update as long as I:012/13 is true.

For Next Loop (FOR, NXT), Break (BRK)

Description:



Use FOR, BRK and NXT instructions to create your own programming routines where you control the number of times the loop is executed.

Important: During prescan, ladder instructions within the FOR/NXT loop are prescanned (not skipped).



ATTENTION: Using FOR and NXT instructions within an output branch can cause unpredictable machine operation.

When using the FOR and NXT instructions within a branch in a ladder program, the execution of the FOR/NXT loop may not occur as expected. Do not use the FOR or NXT instructions when programming within a branch in a ladder program.

Entering Parameters

To program the FOR instruction, you must provide the processor with the following information:

Parameter:	Definition:
Label number	the unique label number that marks the location of the FOR instruction. Enter a unique number. Classic PLC-5 processors support label numbers 0-31; Enhanced PLC-5 processors support label numbers 0-255.
Index	the logical address where the instruction stores the index value it computes. The index value is the sum of the initial value plus the accumulating step values. The FOR instruction uses the index value to determine the number of times the loop is executed. When you enable the FOR instruction, the processor sets the index value equal to the initial value. Then, if the index value is less than or equal to the terminal value, the processor loops through the following instructions. If the index is greater than the terminal value, the processor jumps to the NXT instruction. When the processor encounters a NXT instruction, it returns to the corresponding FOR instruction, then it compares the index with the terminal value. If the index is less than or equal to the terminal value, the processor jumps back to the FOR instruction. Otherwise it steps to the following instruction. If the processor encounters a BRK on a true rung, the processor skips to the instruction following the NXT.
Initial value	(index value) is an integer value or integer address that represents the starting value for the loop.
Terminal value	(reference value) is an integer value or integer address that represents the ending value for the loop.
Step size	(constant) is an integer value that specifies the amount to increment the index value. You can change the step value from the ladder program.

Using FOR

When the rung is true, the FOR instruction executes the rungs between the FOR and NXT repeatedly in one program scan until it reaches the preset number of loops, or a BRK instruction aborts the operation. The FOR instruction repeats this operation each scan its rung is true. The FOR instruction does not require a rung transition to begin operation.

When the rung is false, the processor jumps to the rung following the NXT instruction.

Important: Be careful not to loop too many times in the single program scan. An excessive number of calls causes the watchdog timer to time out, which faults the processor.

You can change the initial and terminal values from the main program before executing the FOR instruction. You should not change the index value.



ATTENTION: Changing the index value could cause the instruction to execute the loop an unexpected number of times with possible damage to equipment and/or injury to personnel.

Also, if you edit a FOR/NXT instruction in Remote RUN mode, make sure that you make the corresponding changes to both rungs before assembling edits. For example, if you want to change the label number for the FOR/NXT pair, change the label in the FOR instruction and in the NXT instruction; then assemble the edits. If you assemble edits after changing only one of the instructions in the FOR/NXT pair, the processor causes a run-time error or watchdog timeout.

Using BRK

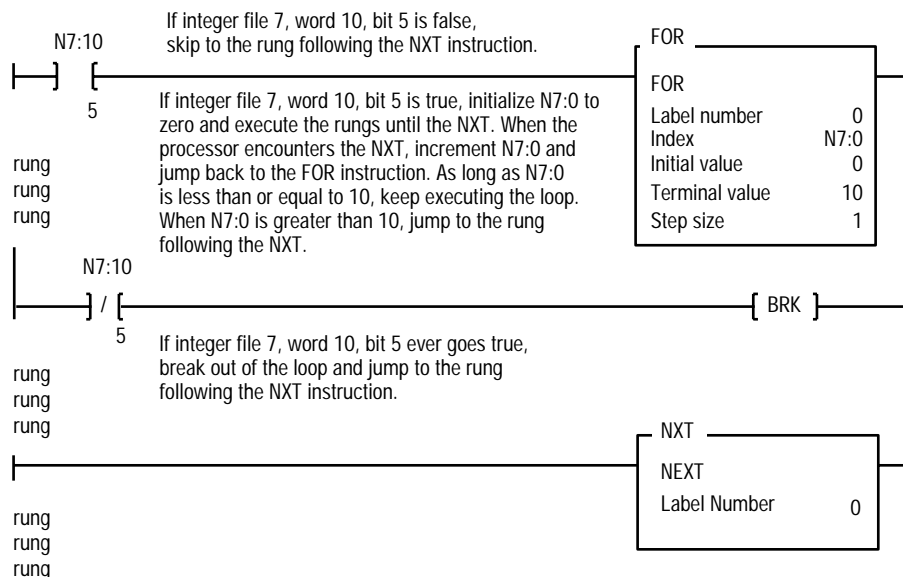
The BRK instruction stops the FOR instruction's operation. Place the BRK rung anywhere between the FOR and NXT rungs. When the rung goes true, it returns the processor to the next highest loop (if you are using nested loops) or to the following instruction after the corresponding NXT instruction in the main program.

Use BRK to exit the loop whenever the processor detects an error or to avoid prolonged loops that could cause the watchdog timer to time out, which would fault the processor.

Using NXT

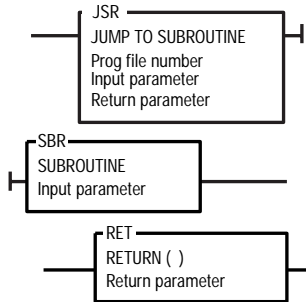
The NXT instruction must be programmed on an unconditional rung that is the last rung to be repeated by the For-Next loop. The NXT instruction returns the processor to the corresponding FOR instruction (identified by the label number specified in the NXT instruction).

FOR, BRK, and NXT Example:



Jump to Subroutine (JSR), Subroutine (SBR), and Return (RET)

Description:



The JSR, SBR, and RET instructions direct the processor to go to a separate subroutine file within the ladder program, scan that subroutine file once, and return to the point of departure.

The JSR instruction directs the processor to the specified subroutine file, and if required, defines the parameters passed to and received from the subroutine. The optional SBR instruction is the header instruction that stores incoming parameters. Use SBR only if you want to pass parameters. The RET instruction ends the subroutine, and if required, stores parameters to be returned to the JSR instruction in the main program.

Important: If you use the SBR instruction, the SBR instruction must be the first instruction on the first rung in the program file that contains the subroutine.

Use a subroutine to store recurring sections of program logic that can be accessed from multiple program files. A subroutine saves memory because you program it only once.

Update critical I/O within subroutines using immediate input and/or output instructions (IIN, IOT), especially if your application calls for nested or relatively long subroutines. Otherwise, the processor does not update I/O until it reaches the end of the main program (after executing all subroutines). Outputs in subroutines are left in their last state.

Passing Parameters

Pass selected values to a subroutine before execution so the subroutine can perform mathematical or logical operations on the data and return the results to the main program.

For example, you can write a generic subroutine for multiple recipe operations. Then pass preset values for each recipe to the subroutine in advance, or have the main program specify and pass presets according to application requirements.

You can pass the following types of parameters:

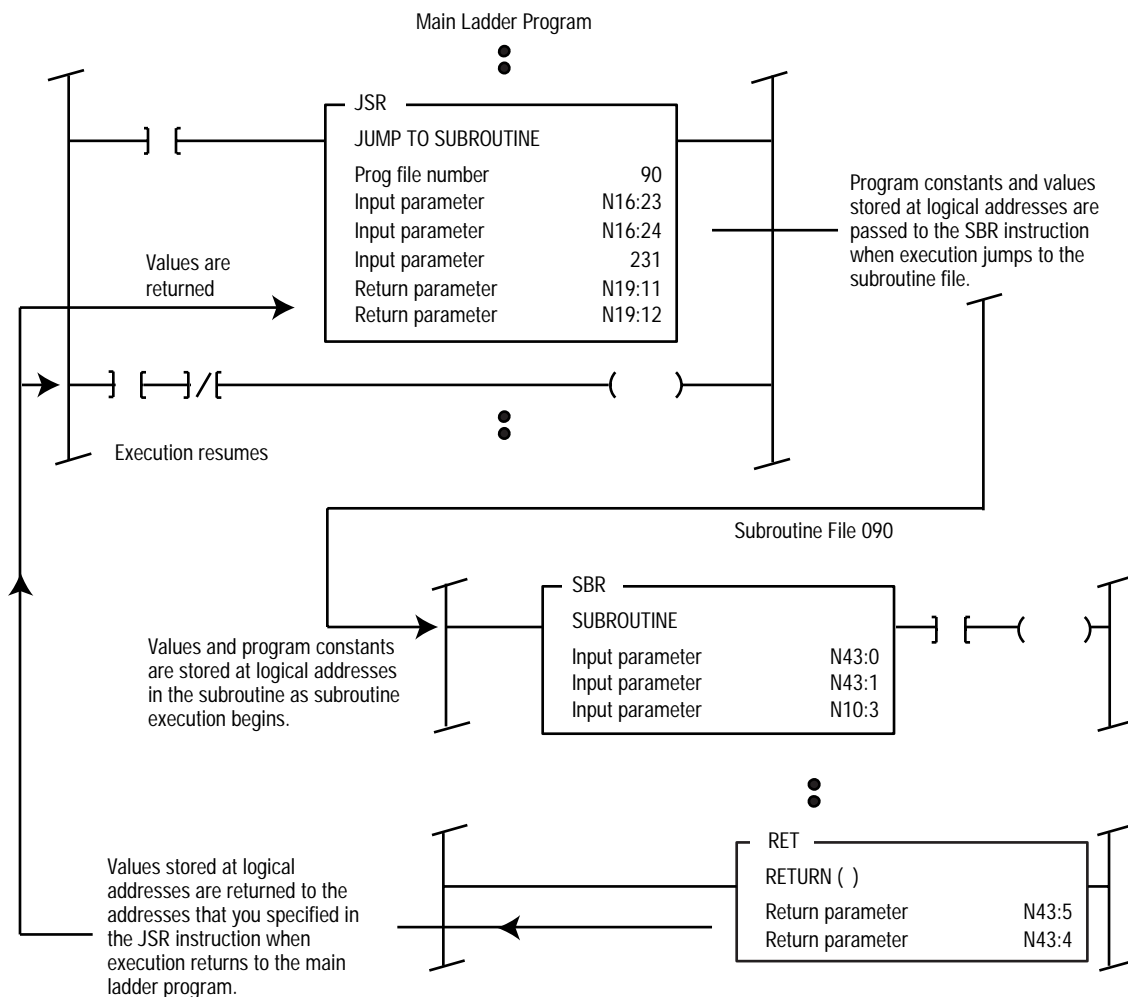
Type:	Example:
Program constant (integer)	256
Program constant (floating point)	23.467
Logical element address	N7:0
Logical structure address	C5:0.ACC

If you pass floating point data to an integer address, the fractional part of the value is truncated (lost).

Important: Do not mix floating point and integer data and addresses when passing data or you will lose accuracy.

Example of Passing Parameters:

The following diagram shows the passing of parameters between a main program file and a subroutine file.



Entering Parameters

To program these instructions, provide the processor with the following information:

Parameter:	Definition:
Program file number	the program file number of the file that contains the subroutine
Input parameter (JSR)	a program constant or an address of a parameter to be sent to the subroutine (optional)
Input parameter (SBR)	an address where the subroutine stores the incoming data (optional)
Return parameter (JSR)	an address that stores the data received from the subroutine (optional)
Return parameter (RET)	a program constant or an address of a parameter to be returned to the JSR instruction in the main program (optional)

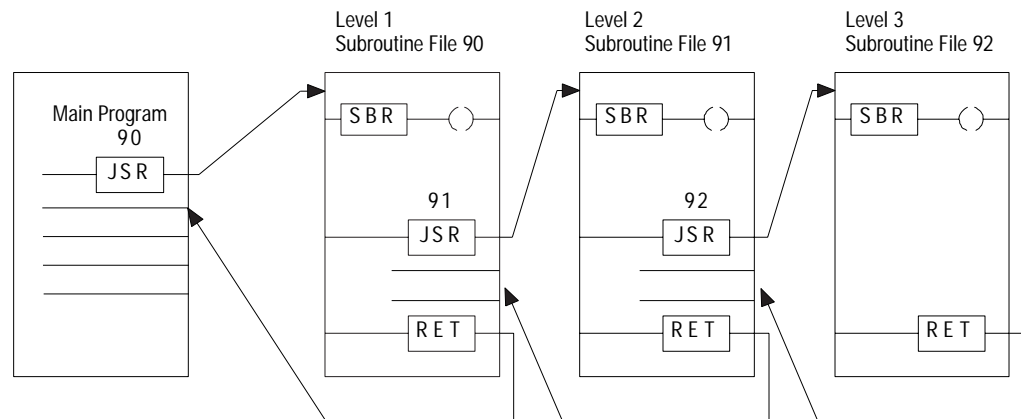
When entering input and return parameters:

- When you enter the JSR instruction, the programming software prompts you for input parameters. After you enter an input parameter, press [**Enter**]. The software prompts you for another input parameter. When you do not have any more input parameters to enter, press [**Enter**] again. Then the programming software prompts you for return parameters in the same manner as it did for the input parameters. You cannot enter more than eight input and return parameters combined.
- Make the number of JSR inputs to your subroutine greater than or equal to the number of input parameter addresses in the SBR instruction. Fewer inputs than addresses to receive them causes a run-time error.
- Make the number of RET return parameters greater than or equal to the number of JSR return addresses to receive them. Fewer outputs than addresses to receive them causes a run-time error.

Nesting Subroutine Files

You can nest up to eight subroutines within a program file. This means you can direct program flow from the main program to a subroutine and on to another subroutine, as long there are no more than 7 levels of subroutines.

The path back is the reverse. At RET, the processor automatically returns to the next instruction after the previous JSR instruction. The processor follows this procedure until it returns to the main program.



15294

Using JSR

The JSR instruction directs the processor to the specified subroutine file, and if required, defines the parameters passed to and received from the subroutine.

When programming the JSR, keep the following in mind:

- Each subroutine that is external to the main program file must have its own file, identified by a unique file descriptor.
- You cannot jump into any part of the subroutine file except for the first (SBR) instruction in that file.
- You can nest up to eight subroutine files.

Using SBR

The optional SBR instruction is the header instruction that stores incoming parameters. Use SBR only if you want to pass parameters. When you pass parameters, the SBR instruction must be the first instruction in the first rung of the subroutine. This rung must also have an output instruction. The SBR instruction stores the program constants and data table values passed from the JSR instruction.

Important: If you use the SBR instruction, the SBR instruction must be the first instruction on the first rung in the program file that contains the subroutine.

Using RET

The RET instruction ends the subroutine, and if required, stores parameters to be returned to the JSR instruction in the main program. The RET instruction concludes subroutine execution. The RET instruction directs the processor back to the instruction following the corresponding JSR instruction. The RET instruction also returns data to the previous subroutine or main program.

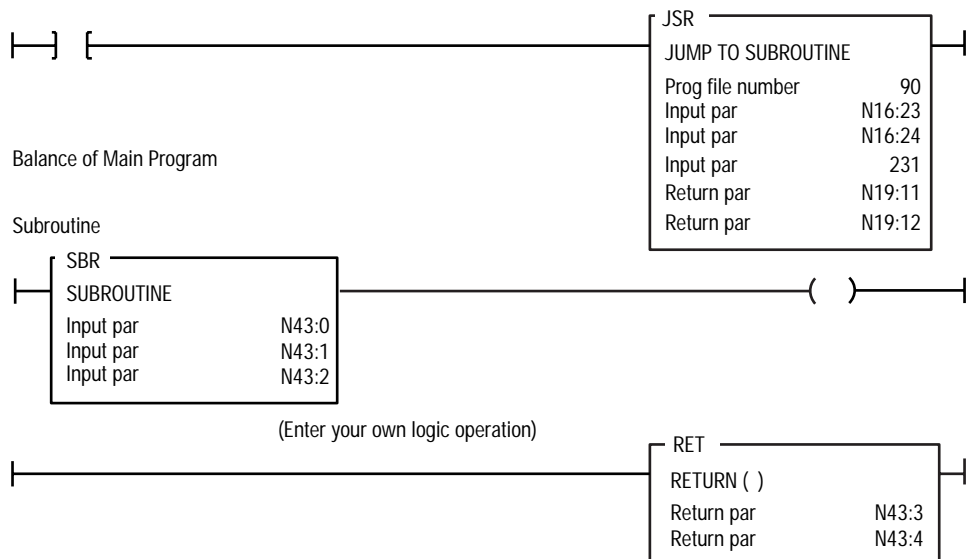
Every subroutine must contain an executable RET instruction if you want to return values from the subroutine. The rung that contains the RET instruction can be conditional. If you use this method, you can program the processor to execute only a part of the subroutine if certain conditions are true. However, be sure to program another RET instruction in an unconditional rung at the end of the subroutine to guarantee a valid return from the subroutine when the conditions on the first RET instruction are false.

Important: To avoid a processor fault, only use the RET instruction in your program when you are returning parameters. If you are not returning parameters, let the end statement in the subroutine do the return to the main program.

JSR, SBR, and RET Example:

When the rung that contains the JSR instruction goes true, the processor jumps to the subroutine file specified by the JSR instruction. The processor also passes three values to the subroutine (value stored at N16:23, value stored at N16:24, and constant 231). Then the processor runs the subroutine logic.

When the processor runs the RET instruction in the subroutine, the processor returns to the instruction following the previous JSR instruction in the main program. The subroutine returns two values to the main program: the value stored at N43:3 is transferred to N19:11, the value stored at N43:4 is transferred to N19:12.



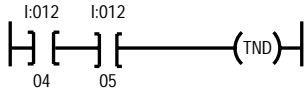
Temporary End (TND)

Description:

When the processor encounters the TND instruction, the processor resets the watchdog timer (to zero), performs an I/O update, and begins running the ladder program at the first instruction in the main program.



Example:



Insert the TND instruction when debugging or troubleshooting your ladder program. The TND instruction lets your program run only up to this instruction. Move it progressively through your program as you debug each new section. Also use the TND instruction as a boundary between the main program and local subroutines. You can program the TND instruction unconditionally, or condition its rung according to your debugging needs.

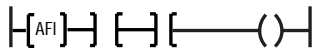
Important: Do not confuse the TND instruction with the end-of-program symbol (EOP). You cannot place instructions on the rung that has the EOP symbol.

Always False (AFI)

Description:


The AFI instruction is an input instruction that makes the rung false when inserted in the condition side of the rung. You can use the AFI instruction to temporarily disable a rung when you debug a program.

Example:



One Shot (ONS)

Description: The ONS instruction is an input instruction that makes the rung true for one program scan upon a false-to-true transitions of the conditions preceding the ONS instruction on the rung.



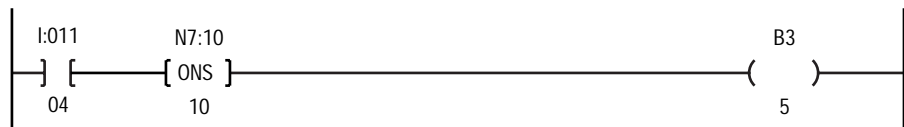
Use the ONS instruction to start events that are triggered by a pushbutton, such as pulling values from thumbwheel switches or freezing rapidly displayed LED values. You must enter a bit address for the bit. Use either a binary file or integer file address. A unique bit must be dedicated to each ONS. You can program an output address for the ONS, but be aware of the following:



ATTENTION: On-line programming with this instruction can be dangerous because the output may turn on immediately when the rung is scanned. Set the bit address value to 1 before entering the instruction; then, the rung must go from false to true before energizing its output.

Important: During prescan, the bit address is set to inhibit false triggering when the program scan begins.

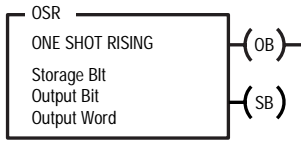
Example:



When the input condition goes from false to true, the ONS conditions the rung so that the output turns on for one scan. The output turns off for successive scans until the input goes from false to true again.

One Shot Rising (OSR) (Enhanced PLC-5 Processors Only)

Description:



The OSR instruction is an output instruction that triggers an event to occur one time. The OSR instruction sets the following bits:

This Bit:	Changes State as Follows:
Output .OB	Is set for one program scan when the rung goes from false to true Note: During prescan, this bit is cleared to inhibit false triggering when the program scan begins.
Storage .SB	Follows the rung status Note: During prescan, this bit is cleared to inhibit false triggering when the program scan begins.

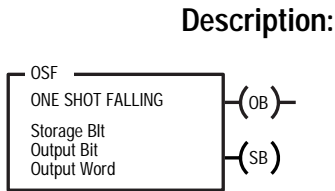
Use the OSR instruction whenever an event must start based on the change of state of the rung from false to true, not continuously when the rung is true. You must enter a bit address for the output bit and storage bit. Use either a binary file or integer file address.

Entering Parameters

To program these instructions, you must provide the processor with the following information:

Parameter:	Definition:
Storage bit	the address where you want the storage bit status stored. For example, B3/17
Output bit	the bit position in the output word where you want the output bit status stored. For example 5
Output word	the word address where you want the output bit status stored. For example, N7:0

One Shot Falling (OSF) (Enhanced PLC-5 Processors Only)



The OSF instruction is an output instruction that triggers an event to occur one time when the rung transitions from true to false. The OSF instruction sets the following bits:

This Bit: **Changes State as Follows:**

Output .OB	Is set for one program scan when the rung goes from true to false
------------	-------------------------------------------------------------------

Storage .SB	Follows the rung status
-------------	-------------------------

Use the OSF instruction whenever an event must start based on the change of state of the rung from true to false, not on the resulting rung status. You must enter a bit address for the output bit and storage bit. Use either a binary file or integer file address.

Entering Parameters

To program these instructions, you must provide the processor with the following information:

Parameter: **Definition:**

Storage bit	the address where you want the storage bit status stored. For example, B3/17
-------------	------------------------------------------------------------------------------

Output bit	the bit position of the output word where you want the output bit status stored. For example 5
------------	------------------------------------------------------------------------------------------------

Output word	the word address where you want the output bit status stored. For example, N7:0
-------------	---------------------------------------------------------------------------------

Sequential Function Chart Reset (SFR) (Enhanced PLC-5 Processors Only)

Description:

SFR	
SFC Reset	
Prog file number	
Restart step at	

The SFR instruction resets the logic in a sequential function chart. When an SFR instruction goes true, the processor performs a postscan/lastscan on all active steps and actions in the selected file, and then resets the logic in the SFC on the next program scan. The chart remains in this reset state until the SFR instruction goes false. The SFR instruction also resets all retentive actions that are currently active.

Entering Parameters

Example:

SFR	
SFC Reset	
Prog file number	2
Restart step at	N7:5

To program this instruction, you must provide the processor with the following information:

Parameter:	Definition:
Program File Number	a valid SFC program file number
Restart Step at	enter one of the following: <ul style="list-style-type: none"> a valid step reference number, 0 to 32767 (entering a 0 defaults to restarting at the initial step) a valid step name an integer address (that stores a step reference number) an address symbol (of an integer address that stores a step reference number)

Important: The Restart Step at parameter is only available with PLC-5/11, -5/20, -5/30 series A, and PLC-5/40, -5/40L, -5/60, -5/60L series B and all Enhanced PLC-5 series C processors. If you are using a PLC-5/40 or -5/60 series A processor, the SFC resets to the initial step.

A step number is a software-assigned reference number associated with each step. You must configure your SFC to display these numbers. For information on configuring your display, see your programming manual.

A step name is any name that you assign to the step. For more information, see the section on assigning step and transition names in your programming manual.

Important: Make sure that the step is actually a step and not a transition or macro. This causes the processor to fault; the software does not perform a check. Also make sure that the step is not within a simultaneous branch or the processor will fault.

Important: Use only one SFR instruction to a single chart. Multiple SFR's in the same chart can produce undesired results since true and false scans of the SFR cause different program behavior.

An analogy would be using multiple TON timer instructions using the same control file. If you want to reset a chart to different positions in the chart based on different conditions, then load the 'step to reset to' into an integer data table location based of the condition and then trigger the SFR.

End of Transition (EOT)

Description:

The EOT instruction should be last instruction in a transition file. If you do not place an EOT instruction in a transition file, the processor always evaluates the transition file as true.

Example:



Important: During prescan, the EOT instruction is skipped so that all ladder instructions can be prescanned.

User Interrupt Disable (UID) (Enhanced PLC-5 Processors Only)

Description: The UID instruction is used to temporarily disable interrupt programs, such as Selectable Timed Interrupts (STI) or Processor Input Interrupts (PII).

—(UID)—

When the rung is true, the UID instruction increments an internal-interrupt-disable counter. As long as this counter value does not equal zero, the currently executing program **cannot** be interrupted by an STI or a PII. Also, if you have a subroutine call within a UIE/UID pair, that subroutine runs without interruption.

The UID instruction does not disable the user fault routine.

Important: Since the UID instruction makes a program un-interruptible, the processor's response time to an STI or PII event may be affected. The UID/UIE section of your program should be as short as possible. Leaving STIs and PIIs disabled for extended periods of time eventually leads to STI and PII overlap errors.

Important: If you have any block transfer in an STI or PII and that block transfer instruction is within a UID/UIE section of your program, the main program scan stops until the block transfer completes.

User Interrupt Enable (UIE) (Enhanced PLC-5 Processors Only)

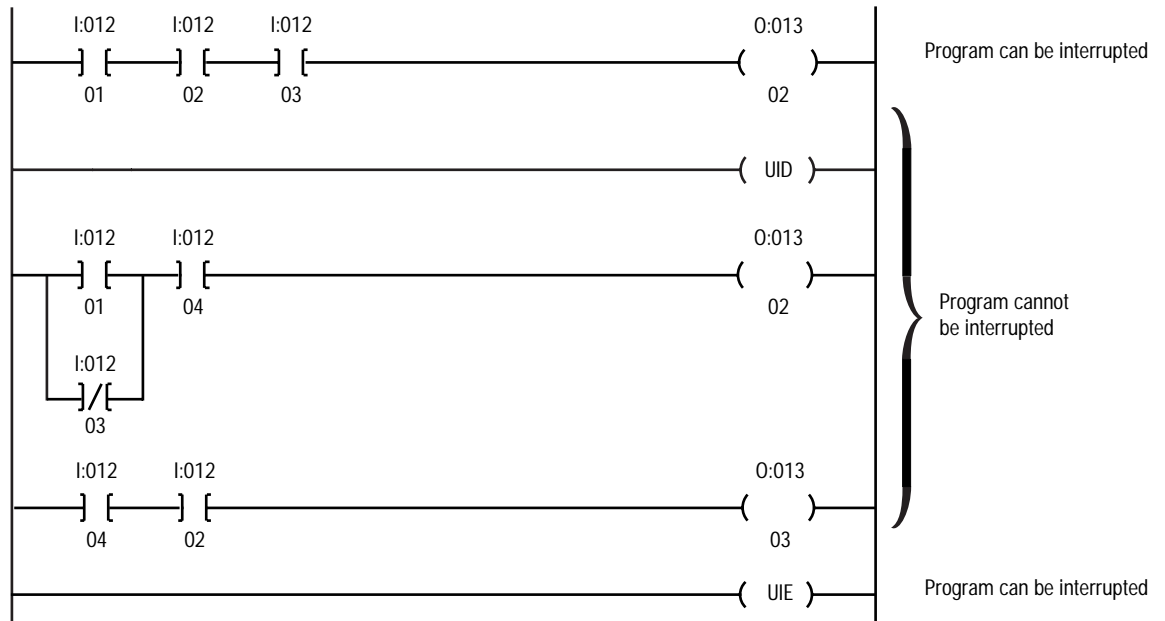
Description: The UIE instruction re-enables STI or PII interrupt programs.

—(UIE)—

When the rung is true and the internal interrupt-disable counter is greater than zero, the interrupt-disable counter is decremented.

When the counter equals zero, the program currently executing is then able to be interrupted again. If there are any interrupt programs pending, they are executed at this time.

Example:

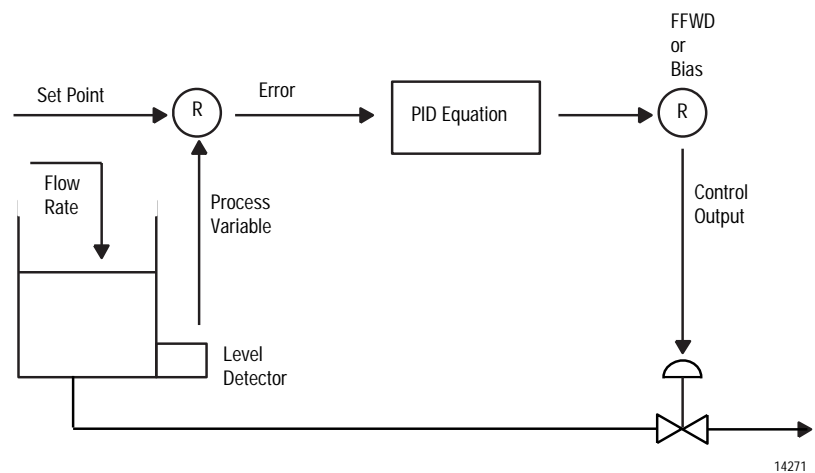


Process Control Instruction PID

Using PID

PID closed-loop control holds a process variable at a desired set point. Figure 14.1 shows a flow-rate/fluid level example.

Figure 14.1
PID Control Example



In the above example, the PID equation controls the process by sending an output signal to the control valve. The greater the error between setpoint and process variable input, the greater the output signal, and vice versa. An additional value (feedforward or bias) can be added to the control output as an offset. The goal of PID calculations is to maintain the process variable you are controlling at the set point.

For programming considerations, see the end of this chapter.

For more information on the operands (and valid data types/values of each operand) used by the PID instruction, see Appendix C.

PID Features

The PID instruction lets the process monitor and control process loops for such quantities as pressure, temperature, flow rate, and fluid level. Features of the PID instruction include:

- PID equations expressed in ISA or Independent Gains
- input and output range from 0-4095 (12-bit analog)
- input scaling in engineering units
- zero-crossing dead band
- derivative term (can act on PV or error)
- direct or reverse acting control
- output alarms
- output limiting with anti-reset windup
- manual mode (with bumpless transfer)
- feedforward or output biasing
- displaying and monitoring PID values

Using PID Equations

The PID instruction has two specific formats, integer control block type and PD control block type. Both formats use the same computational mechanics for the base equation, but differ in options and type of math performed, specifically, integer and floating point math.

The base PID equation used in both cases is the standard parallel position PID Algorithm, with option for entering gains as 'independent' or 'dependent.' The latter option is recognized as ISA standard format.

The processor gives you six choices of PID algorithms, as follows:

Standard equation with dependent gains (ISA standard):

Derivative of Error:

$$CV = K_c \left[(E) + \frac{1}{T_i} \int_0^t (E) dt + T_d \frac{d(E)}{dt} \right] + Bias$$

Derivative of PV:

$$CV = K_c \left[(E) + \frac{1}{T_i} \int_0^t (E) dt + T_d \frac{d(PV)}{dt} \right] + Bias (E = SP - PV)$$

$$CV = K_c \left[(E) + \frac{1}{T_i} \int_0^t (E) dt + T_d \frac{d(PV)}{dt} \right] + Bias (E = PV - SP)$$

Independent gains equation:

Derivative of Error:

$$CV = K_p(E) + K_i \int_0^t (E)dt + K_d \frac{d(E)}{dt} + Bias$$

Derivative of PV:

$$CV = K_p(E) + K_i \int_0^t (E)dt - K_d \frac{d(PV)}{dt} + Bias \quad (E = SP - PV)$$

$$CV = K_p(E) + K_i \int_0^t (E)dt + K_d \frac{d(PV)}{dt} + Bias \quad (E = PV - SP)$$

Where:

K_p	= Proportional Gain (Unitless)	SP	= Set Point
K_i	= Integral Gain (Seconds ⁻¹)	PV	= Process Variable
K_d	= Derivative Gain (Seconds)	Error	= (SP - PV) or (PV - SP)
$\frac{1}{T_1}$	= Reset Gain (Repeats/Minute)	Bias	= Feed-Forward or External Bias
T_d	= Rate Gain (Repeats/Minute)	CV	= Output Control Variable
		Δ_t	= Loop Update Time

Conversion of Gain Constants

Convert from standard to independent gains constants by substituting controller gain (Kc), reset (1/Ti), and rate (Td) values in the following formulas.

$$K_p = K_c \text{ unitless}$$

$$K_i = \frac{K_c}{60T_i} \text{ inverse seconds}$$

$$K_d = K_c(T_d)60 \text{ seconds}$$

Integral Term Implementation

Perform integration by maintaining an accumulated sum, S_k .

In the case of Independent Gains: $S_k = K_i(E_k)\Delta_t + S_{k-1}$

With Dependent Gains selected: $S_k = \frac{1}{T_i}(E_k)\Delta_t + S_{k-1}$

If the Integral or Reset Gain is zero, the accumulated sum continually sets to zero in Auto mode.

Avoid 'Integral Wind-up' by preventing the running sum from accumulating whenever the output (CV) reaches its maximum or minimum values. These values are either 0% and 100% or the user specified limits in output limiting. In this case, $S_k = S_{k-1}$.

The accumulated sum remains 'frozen' until the output drops below its maximum value or rises above its minimum value; then normal accumulation resumes.

When executing the PID instruction in manual mode, a 'bumpless' transfer back to Auto mode can be achieved by using the accumulated sum to computationally track the manual output:

$$= CV_{Manual} - Bias - K_p(E) - K_d \frac{d(I)}{dt}$$

When you switch back to auto mode, the PID computation yields this Manual Output value and no 'jump' in output occurs as a result of the mode change.

Derivative Term

The following approximation is used to calculate the derivative term:

$$\frac{d(Q)}{dt} = \frac{Q_k - Q_{k-1}}{\Delta_t}$$

Where Q represents either Error or PV,
depending upon your settings.

The calculation is further enhanced by using a 'derivative smoothing filter.' This first order, low pass, digital filter eliminates large derivative term 'spikes' caused by noise in the PV.

Adding this filter to the overall derivative term yields:

$$D_k = (1 - \alpha) \left[K_d \frac{Q_k - Q_{k-1}}{\Delta_t} \right] + \alpha D_{k-1}$$

Where:

$$\begin{aligned} K_d &= \text{the derivative gain} \\ D_k &= \text{the current derivative term} \\ D_{k-1} &= \text{the previous derivative term} \\ Q_k &= \text{(as previously defined)} \\ \alpha &= \frac{1}{16 \frac{\Delta_t}{K_d} + 1} \\ \Delta_t &= \text{Loop Update Time} \end{aligned}$$

Setting Input/Output Ranges

The input module that measures the processor variable (PV) must have a full-scale binary range of 0-4095. The processor ignores the upper four most-significant bits of the 16-bit process variable (integer PID only).

The control output has the same range of 0-4095. You can set limits on the output to limit the output calculated by the PID instruction to any value in the range of 0-4095.

The tieback input (output tracking) from a manual control station must also have a range of 0-4095. The PID instruction uses the result to calculate the integral accumulated value, which allows for bumpless transfer from manual to automatic control.

The PID instruction also copies the tieback value into the control output storage location when in manual mode. Tieback input is only used when you use a hardware auto/manual station. Otherwise, set the tieback value to zero.

Implementing Scaling to Engineering Units – Integer File Type

You can scale the setpoint and zero-crossing dead band values to engineering units for integer file types. You can also display the process variable and error values in these same engineering units.

When you select scaling, the PID instruction scales the setpoint, dead band, process variable, and error values. You also have:

1. Enter the maximum and minimum values S_{max} and S_{min} in the PID control block (words 7 and 8). The S_{min} value corresponds to an analog value of zero for the lowest reading of the process variable; the S_{max} value corresponds to an analog value of 4095 for the highest reading of the process variable. These values represent process limits. Set S_{min} and S_{max} to 0 if you do not want scaling.

For example, if you measure a scale of temperature from -73 (PV=0) to $+1156$ (PV=4095), enter -73 for S_{min} and 1156 for S_{max} .

If the analog input module is not configured to return a value in the range 0-4095, see “Descaling Inputs” on page 14-27 in this chapter.

2. Reset bit 5 of word 0 in the PID control block (integer file type only). Set this bit only if you want to inhibit scaling the setpoint. You must inhibit setpoint scaling of a cascaded inner loop while scanning other loop variables.

3. Enter the setpoint, word 2, and dead band, word 9 (integer file type only), values in the same scaled engineering units. The control output (word 16) displays as a percentage of the 0-4095 range. The output the processor transfers to the output module is always unscaled.

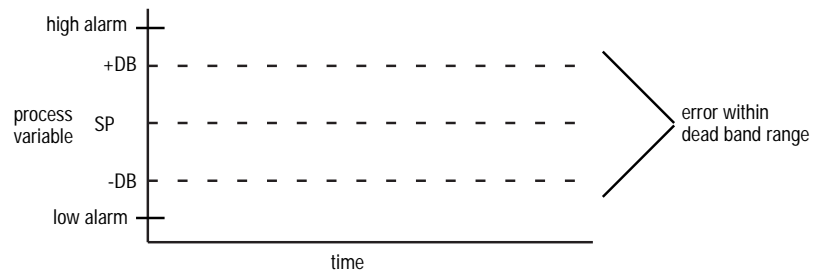


ATTENTION: Do not change scaling when the processor is in Run mode. The processor could fault and cause an undesirable process response, possible equipment damage, and personal injury.

Setting the Dead Band

The adjustable dead band 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 dead band lets you control how closely the process variable matches the setpoint without changing the output.



Using Zero-Crossing

Zero-crossing is dead band control that lets the instruction use the error for computational purposes as the process variable crosses into the dead band 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 dead band, the instruction considers the error value zero.

Enter your dead band value in word 9 of the control block (.DB word of a PD data file type). The dead band extends above and below the setpoint by the value you specify. Enter 0 to inhibit the dead band. If scaled, the dead band has the same scaled units as the setpoint.

Using No Zero Crossing

The series E processor added a no zero crossing feature, which is useful for applications that run high inertia processes that slowly move a high mass which is hard to stop. The no zero crossing feature causes the CV output not to change value as long as the PV is inside the deadband range, instead of only after reaching the setpoint value. With the proper tuning, it is then possible to have the PV drift to the setpoint value.

Selecting the Derivative Term (Acts on PV or Error)

Derivative is a change of state variable. You can select whether the derivative term in either PID equation acts on changes in the processor variable or error value. Use bit 6 of word 0 in the control block (.DO word of a PD data file type) to select the type of derivative action you want.

Setting Output Alarms

You can set an output alarm on the control variable output at a selected value above or below the setpoint. When the instruction detects that the output has reached either value, the processor sets an alarm bit (bit 10 for lower limit, bit 9 for upper limit) in word 0 of the control block (.OLH and .OLL bits of a PD data file type). Alarm bits are reset by the instruction when the output comes back inside the limits. The instruction does not prevent the output from exceeding the alarm values unless you select output limiting.

Enter the upper output alarm in word 11 (.MAXO) and the lower output alarm in word 12 (.MINO) of the control block. The processor handles output alarm values as a percentage of the output. If you do not want alarms, enter 0% for the lower alarm and 100% for the upper alarm.

Using Output Limiting

You can set an output limit (percent of output) on the control output. When the instruction detects that the output has reached a limit, it sets an alarm bit (bit 10 .OLL for the lower limit, bit 9 .OLH for the upper limit) in word 0 of the control block and prevents the output from exceeding either value. The instruction limits the output to 0 and 4095 if you do not specify a limit.

To use output limits, set the limit enable bit (bit 03 of word 0) and enter the upper limit in word 11 and the lower limit in word 12. Limit values are a percentage (0-100%) of the output.

Important: If you are using the PD data file type for the control block, the processor performs this function without you having to set bits.

Anti-Reset Windup

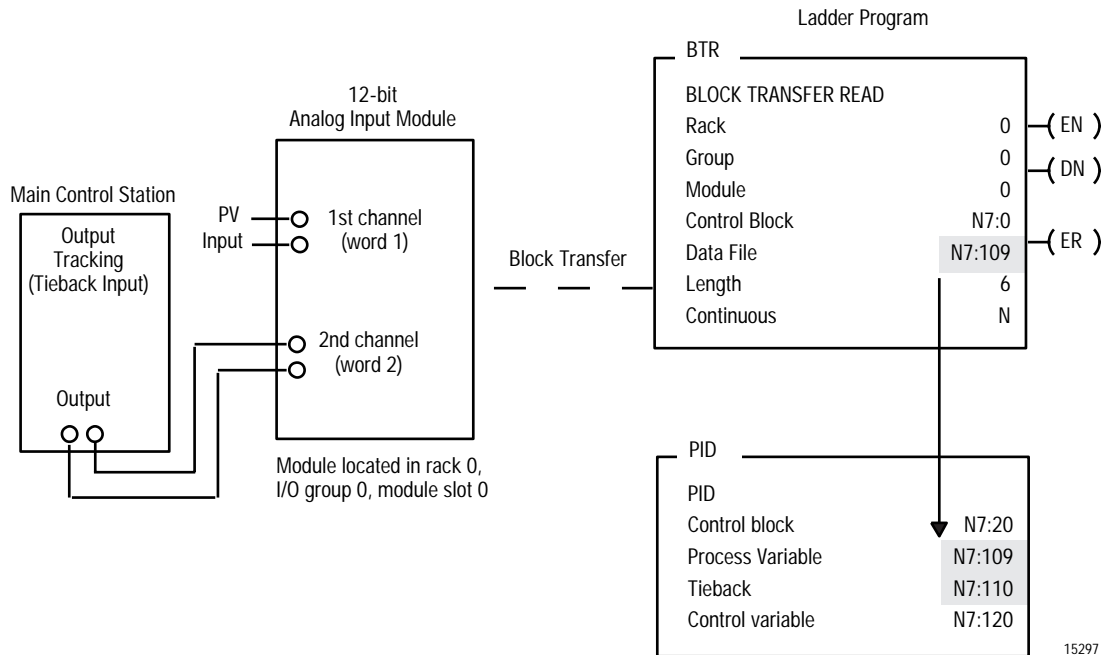
Anti-reset windup is a feature that prevents the integral term from becoming excessive when outputs reach a limit. When the sum of the PID and bias terms in the output reaches a limit, the instruction stops calculating the integral output term until the output comes back into range.

Using a Manual Mode Operation (with Bumpless Transfer)

Manual operation lets an output from a manual control station or from your ladder program override the calculated output of the PID instruction.

With a manual control station, you control the output device directly and override the PID instruction's output. You must feed the output value into the PID instruction's tieback input (Figure 14.2). The PID instruction uses this value to calculate the integral term value required to achieve a bumpless transfer when you switch from manual to automatic control.

Figure 14.2
Example Diagram for Moving Analog Inputs to a PID Instruction



15297

Set Output

You can replace a manual control station with a thumbwheel and pushbutton switches, and simulate the PID function with ladder logic.

Use the Set Output mode to enter a value representing a percentage of the Control Variable output. Typically, it is desired to enter a value from an operator interface. The table below illustrates the procedure to follow when Set Output mode is desired.

Table 14.A
Set Output Mode Procedure

	Integer Control Block (N7:0)	PD Control Block (PD10:0)
Select Automatic Mode	Mode:0 (0:auto/1:manual) (bit N7:0/1 = 0)	A/M Station Mode = Auto (bit PD10:0.MO = 0)
Select Set Output Mode	SET OUTPUT MODE: 1 (0:no/1:yes) (bit N7:0/4 = 1)	Software A/M Mode = Manual (bit PD10:0.SWM = 1) Note: In data monitor, MODE=AUTO changes to MODE=SW MANUAL.
Enter % in Set Output Value (0-100%)	SET OUTPUT VALUE % (word N7:10 = %value)	SET OUTPUT % (word PD10:0.SO = % value)

If the set output value is greater than the upper CV limit or lower than the lower CV limit and output limiting is enabled and the PID instruction is in Set Output mode, the processor uses the actual output (not the set output value) to calculate the integral accumulator term for calculating bumpless transfer.

Feedforward or Output Biasing

You can feedforward a disturbance from the system or bias output by feeding either of these values into the PID instruction's feedforward/bias word (word 6 PD.BIAS) of the control block. Either value must have a range of -4095 to $+4095$ (integer) or -100.0% to $+100.0\%$ (floating point).

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 faster than waiting for the process variable to change as a result of mixing.

A bias value can be used to compensate for a steady-state loss of energy from the controlled process.

Resume Last State

With the resume-last-state function, you can make full use of the analog output module's hold-last-state function. The resume-last-state function lets the PID instruction resume calculating the integral term of the PID algorithm from its last output value (instead of zero) when returning to Run mode.

If you are using an integer data file for the control block, set the bits according to the guidelines below. If you are using a PD data file type for the control block, the processor saves the integral accumulator and uses it when going from Program to Run mode.

Use this function as follows:

- Set word 0, bit 7 if you configured the analog output module to hold last state if a fault occurs and when changing from Run to Program mode
- Reset word 0, bit 7 if you configured the analog output module to turn off if a fault occurs and when changing from Run to Program mode



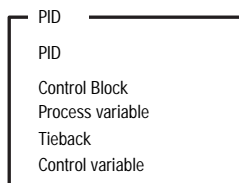
ATTENTION: If you want to use this function, set bit 7 only after the PID instruction has executed at least once (at start-up) or when returning to Run mode. If you do not let the PID instruction execute at least once, unpredictable machine operation may occur causing possible damage to equipment and/or injury to personnel.

Resume Last State is available with the following processors:

- Enhanced PLC-5 processors all series/all revs
- PLC-5/12 series A/rev C and later
- PLC-5/15 series B/rev H and later
- PLC-5/25 series A/rev D and later

PID Instruction

Description:



The PID instruction is an output instruction that controls physical properties, such as temperature, pressure, liquid level, or flow rate of process loops.

The PID instruction controls a PID loop with inputs from an analog input module and an output to an analog output module. For temperature control, you can convert the analog output to a time proportioning on/off output for driving a heater or cooling unit.

Execute the PID instruction periodically at constant intervals using a timer, a selectable timed interrupt (STI), or real-time sampling. The ladder program can interact with the PID algorithm by changing variables during operation, or you can change variables from a programming terminal or from stations on a Data Highway™ or Data Highway Plus™ communications link.

The PID instruction provides bumpless transfer even when not using the integral gain. It does this by generating a bias term equal to the difference between the proportional term and the manually adjusted output as follows:

If you select manual mode with tieback:

$$\text{BIAS} = (\text{TIEBACK} - \text{Pterm}) - \text{Dterm}$$

If you select manual mode with setoutput:

$$\text{BIAS} = (\text{SETOUTPUT mode} - \text{Pterm}) - \text{Dterm}$$

Normally, the processor reads the bias term value you specify in the PID configuration block. However, under one condition, the processor will write a value to the bias term. This occurs when the integral gain equals zero and the mode of the loop is changed from manual to auto. The processor back calculates the integral accumulator in an attempt to provide a bumpless transfer when going from manual to auto.

The bumpless transfer function is available with the following (or greater) processor revision levels:

- Enhanced PLC-5 processors, all series, all revisions
- PLC-5/12 series A revision C
- PLC-5/15 series B revision H
- PLC-5/25 series A revision D

Processors with earlier revision levels provided bumpless transfer only when the integral term was included in the PID algorithm.

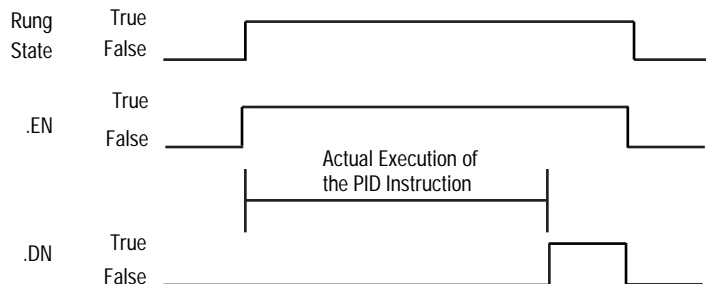
Using No Back Calculation

The no back calculation feature is for applications where you do not want the bias value for the CV output to be overwritten when in manual or set output (software manual) mode. When you select no back calculation and the mode is either of the manual modes and the integral gain is zero, the PID instruction does not perform the back calculation into the bias term. In this condition, a bump could occur in the CV output.

Operational Status Bits

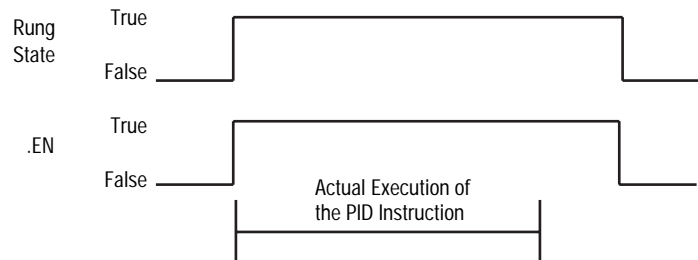
Integer Block

The Integer Block PID instruction uses an enable bit (.EN) to indicate its qualifying rung conditions made a false-to-true transition. The rung conditions have remained true, indicating the enable bit is true. The only way that the enable bit becomes false again is if those same qualifying conditions become false or the enable bit is purposely “unlatched” through ladder logic. The Integer Block done bit (.DN) becomes true when the PID instruction successfully completes execution and remains true until the qualifying rung conditions become false.



PD Block

The PD Block PID instruction has only an enable bit (.EN) to indicate operational status. This bit indicates that its qualifying rung conditions are true, in which case the enable bit is true (a false-to-true transition is not needed). The only way the enable bit becomes false again is if these same qualifying conditions become false. The PD block does not use a done bit.



Important: Unlike the Integer Block version, the PD Block PID executes again if the program scan encounters this rung again while the rung state is still true.

Entering Parameters

When you enter the instruction, you must specify four addresses that are fundamental to the operation of the instruction. After you enter these addresses, the programming software displays a screen from which you enter the operating parameters of the instruction.

The use of integer control blocks versus PD control blocks depends on your processor. If you are using a Classic PLC-5 processor, the PD control block is not available. In the Enhanced PLC-5 processors, both the N and PD control blocks are available. The PD control block gives you more flexibility (i.e., floating point variables, higher resolution – 12 bit versus 16 bit).

The addresses that you enter are:

Parameter:	Definition:
Control Block	<p>a file that stores PID status and control bits, constants, variables, and internally used parameters.</p> <p>Based on the data type you use, a different configuration screen appears for you to enter PID information (see the next sections for more information).</p> <p>If you have an Enhanced PLC-5 processor, you can use an integer control block or a PD control block. Using a PD file, words 0, 1 are the status words; words 2-80 store the PID values.</p> <p>If you use an Integer control block, the PID calculations are performed using integer values. If you use a PD control block, the PID calculations are performed using floating point values.</p> <p>If you have a Classic PLC-5 processor, you must use an integer file (N) for your control block. Using an integer file, word 0 is the status word; words 1-22 store the PID values.</p>
Process Variable	a word address that stores the process input value.
Tieback	a word address used to implement bumpless transfer when using a manual control station. The tieback is an output of a BTR instruction from the station.
Control Variable	<p>a word address to which the PID instruction sends its calculated PID output value.</p> <p>Note: If a value greater than 4095 is written to the "control variable" location of the Integer Type PID instruction, the output of the PID instruction obtains a permanent offset which can only be removed by writing to the "control variable" with a value between 0 and 4095. This happens whether you write to this location via rung logic or directly to the data table location.</p> <p>Note: The file PD type PID instruction does not exhibit this behavior.</p>

Using an Integer Data File Type for the Control Block

When using an integer data file type for the control block, the data monitor screen for the PID instruction shows the following information, some of which is display only; some of which you specify the values (Table 14.B).

Table 14.B
PID Parameter Descriptions (Integer Control Block)

Parameter:	Description:
Equation	<p>Enter whether you want to use independent (0) or dependent (1) gains. Displays one of the following:</p> <p>INDEPENDENT (0) – for independent gains DEPENDENT (1) – for dependent gains (ISA)</p> <p>Use dependent gains when you want to use standard loop tuning methods. Use independent gains when you want the three gain constants (P, I, and D) to operate independently.</p>
Mode	<p>Displays operating mode:</p> <p>AUTO (0) – automatic PID control MANUAL (1) – control from a manual control station</p> <p>Sets the use of the tieback parameter for manual operation</p>
Error	<p>Displays one of the following error value:</p> <p>Reverse acting: 0 = SP-PV Direct acting: 1 = PV-SP</p>
Output Limiting	<p>Displays whether or not the instruction clamps the output at the high and low limiting values. Displays one of the following:</p> <p>NO (0) – output not clamped YES (1) – output clamped</p> <p>The PID algorithm has an anti-reset windup feature that prevents the integral term from becoming too large when the output reaches the high or low alarm limits. If the limits are reached, the algorithm stops calculating the integral term until the output comes back into range.</p>
Set output mode	<p>Selects the use of set output value % for manual operation</p>
Setpoint scaling	<p>Selects if the setpoint is to be interpreted as a value in the engineering units or an unscaled (0 to 4095) value</p>
Derivative input	<p>Selects if derivative term is based on changes in PV or on changes in error</p>
Last state resume	<p>Selects if you want to resume last state or hold last state</p>

(Continued)

Parameter:	Description:
Deadband status	Set if the PV is inside the selected deadband range; reset if it is not
Upper CV limit alarm	Set if calculated CV is greater than the CV upper limit word %
Lower CV limit alarm	Set if calculated CV is less than the CV lower limit word %
Setpoint out of range	<p>Displays whether or not the setpoint is out of the range of engineering units you selected on the PID Configuration screen. Displays one of the following:</p> <p>NO (0) – SP within range YES (1) – SP out of range</p> <p>Note: A major processor fault occurs if the SP is out of range when the instruction is first enabled.</p>
PID done	Displays whether the PID instruction has completed (1 = done; 0 = not done).
PID enabled	Displays whether the PID instruction is enabled (1 = enabled; 0 = not enabled).
Feed forward	<p>Enter a value between –4095 and 4095 for the amount of feed forward.</p> <p>The ladder program can enter a feedforward value to bump the output in anticipation of a disturbance. This value is often used to control a process with transportation lag.</p>
Max scaled input	Enter the integer number (–32,768 - 32,767) that is the maximum value available from the analog module. For example, use 4095 for a module whose range is 0-4095.
Min scaled input	Enter the number that is the minimum value available from the analog module. For example, use 0 for a module whose range is 0 to 4095.
Dead band	<p>For an unscaled deadband, enter a value in the engineering units you selected on the PID Configuration screen. Valid range is 0 to 4095 unscaled, –32,768 to +32,767 scaled.</p> <p>Note: The deadband is zero crossing.</p>
Set output value %	Enter a percent (0-100%) to use as the CV Output when 'set output mode' is selected.
Upper CV limit %	Enter a percent (0-100) above which the algorithm clamps the output.
Lower CV limit %	Enter a percent (0-100) below which the algorithm clamps the output.
Scaled PV value	Displays data from the analog input module that the instruction scales to the same engineering units that you selected for the setpoint.
Scaled error	Displays the current error in scaled engineering units
Current CV %	Displays current controlled variable output value as a percent
Setpoint	Enter an integer. Valid range is 0 to 4095 (unscaled) or Smin- Smax (scaled engineering units).
Proportional gain (K _c)	Enter an integer. Valid entry range is 0-32,767 (unitless) or K _p 0-32,767. The processor divides the entry value by 100 for calculations.
Reset time (T _i) minutes/repeat	<p>Enter an integer. Valid entry range for T_i is 0-32,767 (minutes times 100). The processor automatically divides the entry value by 100 for calculations.</p> <p>Valid entry range for K_i is 0-32,767 (inverse seconds times 1000). The processor automatically divides the entry by 1000 for calculations.</p>

(Continued)

Parameter:	Description:
Derivative rate (T_d)	Enter an integer. Valid entry range is 0-32,767 or KD 0-32,767. The processor divides the entry value by 100 for calculations.
Loop update time	<p>Enter an update time (greater than or equal to 0.01 seconds) at 1/5 to 1/10 times the natural period of the load (load time constant). Valid entry range is 1-32,767 seconds. The processor divides the entry value by 100 for calculations. The load time constant should be greater than:</p> <p style="text-align: center;">$1ms(\text{algorithm}) + \text{block transfer time (ms)}$</p> <p>Periodically enable the PID instruction at a constant interval equal to the update time. For update times of less than 100 msec, use an STI. When update times are greater than 100 msec, use a timer or a real-time sampling.</p> <p>Note: If you omit an update time or enter a negative update time, a major fault occurs the first time the processor runs the PID instruction.</p>

Using Control Block Values

Word 0 of the control block contains status and control bits. Table 14.C shows the values stored in each word of the control block.

Table 14.C
PID Control Block (Integer Control Block)

Word:	Contains:	Term:	Entry Range:
0	Bit 15 Enabled (EN) Bit 13 Done (DN) Bit 11 Set point out of range Bit 10 Output alarm, lower limit Bit 9 Output alarm, upper limit Bit 8 DB, set when error is in deadband Bit 7 Resume last state (0=yes; 1=hold last state) Bit 6 Derivative action (0=PV, 1=error) Bit 5 Setpoint descaling (0=no, 1=yes) Bit 4 Set output (0=no, 1=yes) Bit 3 Output limiting (0=no, 1=yes) Bit 2 Control (0=reverse, 1=direct) Bit 1 Mode (0=automatic, 1=manual) Bit 0 Equation (0=independent, 1=ISA)		
1	reserved		
2	Setpoint	SP	0-4095 (unscaled) Smin-Smax (scaled)

Note: Terms marked with an asterisk (*) are entered as $Y_y \times 100$. The term itself is Y_y . The term marked with a double asterisk (**) is entered as $Y_y \times 1000$. The term itself is Y_y .

(Continued)

Word:	Contains:	Term:	Entry Range:
3	Independent:	Proportional gain x 100 (unitless)	K_p^* 0-32,767
	ISA:	Controller gain x 100 (unitless)	K_c^* 0-32,767
4	Independent:	Integral gain x 1000 (1/sec)	K_i^{**} 0-32,767
	ISA:	Reset term x 100 (minutes per repeat)	T_i^* 0-32,767
5	Independent:	Derivative gain x 100 (seconds)	K_d^* 0-32,767
	ISA:	Rate term x100 (minutes)	T_d^* 0-32,767
6	Feedforward or bias	FF/Bias	-4095-+4095
7	Maximum scaling	Smax	-32,768-+32,767
8	Minimum scaling	Smin	-32,768-+32,767
9	Dead band	DB	0-4095 (unscaled) $S_{min}-S_{max}$ (scaled)
10	Set output	SETOUT	0-100%
11	Maximum output limit (% of output)	Lmax	0-100%
12	Minimum output limit (% of output)	Lmin	0-100%
13	Loop update time x 100 (seconds)	dt	0-32,767
14	Scaled PV value (displayed)		$S_{min}-S_{max}$
15	Scaled error value (displayed)		$S_{min}-S_{max}$
16	Output (% of 4095)	CV	0-100%
17-22	internal storage; do not use		

Note: Terms marked with an asterisk (*) are entered as $Y_y \times 100$. The term itself is Y_y . The term marked with a double asterisk (**) is entered as $Y_y \times 1000$. The term itself is Y_y .

Using a PD File Type for the Control Block (Enhanced PLC-5 Processors Only)

When using a PD file type for the control block, the data monitor screen for the PID instruction shows the following information, some of which is display only; some of which you specify the values (Table 14.D).

Table 14.D
PID Parameter Descriptions (PD Control Block)

Parameter	Address Mnemonic:	Description:
Setpoint	.SP	Enter a floating-point number in the same engineering units that are on the PID Configuration screen. Valid range is $-3.4 \text{ E}+38$ to $+3.4 \text{ E}+38$.
Process Variable	.PV	Displays data from the analog input module that the instruction scales to the same engineering units that you selected for the setpoint.
Error	.ERR	Displays one of the following error values: Reverse acting: Error = PV-SP Direct acting: Error = SP-PV
Output %	.OUT	Displays the PID algorithm control output value (0-100%).
Mode	.MO .MO=0 .MO=1 .SWM=1	Displays operating mode: AUTO – automatic PID control MANUAL – control from a manual control station SW MANUAL – simulated manual control from the data monitor or ladder program
PV Alarm	.PVHA=1 .PVLA=1	Displays whether the PV is within or exceeds the high or low alarm limits you selected on the PID Configuration screen. Displays one of the following: NONE – PV within alarm limits HIGH – PV exceeds high alarm limit (used with deadband) LOW – PV exceeds low alarm limit (used with deadband)
Deviation Alarm	.DVPA=1 .DVNA=1	Displays whether the error is within or exceeds the high or low deviation alarms you selected on the PID Configuration screen. Displays one of the following: NONE – error within deviation alarm limits POSITIVE – error exceeds high alarm (used with deadband) NEGATIVE – error exceeds low alarm (used with deadband)
Output Limiting	.OLH=1 .OLL=1	Displays whether or not the instruction clamps the output at the high and low limiting values (.MAXO and .MINO) you selected on the PID Configuration screen. Displays one of the following: NONE – output not clamped HIGH – output clamped at the high end (.MAXO) LOW – output clamped at the low end (.MINO)

The PID algorithm has a anti-reset-windup feature that prevents the integral term from becoming too large when the output reaches the high or low alarm limits. If the limits are reached, the algorithm stops calculating the integral term until the output comes back into range.

(Continued)

Parameter	Address Mnemonic:	Description:
SP Out of Range	.SPOR=0 .SPOR=1	<p>Displays whether or not the setpoint is out of the range of engineering units you selected on the PID Configuration screen. Displays one of the following:</p> <p>NO – SP within range YES – SP out of range</p> <p>Note: A major processor fault occurs if the SP is out of range when the instruction is first enabled.</p>
Error Within DB	.EWD=0 .EWD=1	<p>Displays whether the error is within or exceeds the deadband value you enter on this screen. The deadband is zero crossing. Displays one of the following:</p> <p>RESET – Error exits the deadband zone SET – Error crosses the deadband centerline</p>
PID Initialized	.INI=0 .INI=1	<p>Each time you change a value in the control block, the PID instruction takes over twice as long to execute (until initialized) on the first scan. Displays one of the following:</p> <p>NO – PID instruction not initialized after you changed control block values YES – PID instruction remains initialized because you did not change any control block values</p> <p>Attention: Do not change the range of input or engineering units when running. If you must do this, then you must reset this bit to re-initialize. Otherwise, the instruction will malfunction with possible damage to equipment and injury to personnel.</p>
A/M Station Mode	.MO=0 .MO=1	<p>Enter whether you want automatic (0) or manual (1) PID control. Displays one of the following:</p> <p>AUTO (0) – automatic PID control MANUAL (1) – manual PID control</p> <p>Manual control specified that an output from a manual control station overrides the calculated output of the PID algorithm.</p> <p>Note: Manual overrides Set Output Mode.</p>
Software A/M Mode	.SWM=0 .SWM=1	<p>Enter whether you want automatic PID (0) control or Set Output Mode (1), for software-simulated control. Displays one of the following:</p> <p>AUTO (0) – automatic PID control SW MANUAL (1) – software-simulated PID control</p> <p>You can simulate a manual control station with the data monitor when you program a single loop. To do this, set .SWM to SW MANUAL and enter a Set Output Percent value.</p> <p>You can simulate a manual control station with ladder logic, pushwheels, and pushbutton switches when you program several loops. To do this, set .SWM to SW MANUAL and move a value into the set output element .SO.</p>

(Continued)

Parameter	Address Mnemonic:	Description:
Status Enable	.EN=0 .EN=1	Enter whether to use (1) or inhibit (0) this bit which displays the rung condition so you can see whether the PID instruction is operating. Displays one of the following: 0 – instruction not executing 1 – instruction executing
Proportional Gain	.KP	Enter a floating-point value. Valid range for independent or standard gains is 0 to 3.4 E^{+38} (unitless).
Integral Gain	.KI	Enter a floating-point value. Valid range for independent gains is 0 to 3.4 E^{+38} inverse seconds; valid range for standard gains is 0 to 3.4 E^{+38} minutes per repeat.
Derivative Gain	.KD	Enter a floating-point value. Valid range for independent gains is 0 to 3.4 E^{+38} seconds; valid range for standard gains is 0 to 3.4 E^{+38} minutes.
Output Bias %	.BIAS	Enter a value (–100 to +100) to represent the percentage of output you want to feed forward or use as a bias to the output. The bias value can compensate for steady-loss of energy from the system. The ladder program can enter a feedforward value to bump the output in anticipation of a disturbance. This value is often used to control a process with transportation lag.
Tieback %	.TIE	Displays a number (0 to 100) representing the percent of raw tieback (0 – 4095) from the manual control station. The PID algorithm uses this number to achieve bumpless transfer when switching from manual to auto mode.
Set Output %	.SO	Enter a percent (0 to 100), from this screen or a ladder program, to represent the software-manually controlled output. When you select software-simulated control (.SWM=1), the PID instruction overrides the algorithm with the set output value (0 - 4095) for transfer to the output module, and copies it to .OUT for display as a percent. The transfer to software-simulated control is bumpless because .SO (under your control) starts with the last automatic algorithm output. Do not vary .SO until after the transfer. To achieve bumpless transfer when changing from software-simulated control to automatic control, the PID algorithm changes the integral term so that the output is equal to the set output value.

When using a PD file type for the control block, the data monitor screen for the PID instruction provides access to a PID configuration screen. From the PID configuration screen, you can define the following characteristics of the PID instruction (Table 14.E).

**Table 14.E
PID Configuration Descriptions (PD Control Block)**

Parameter:	Address Mnemonic:	Description:
PID Equation	.PE=0 .PE=1	<p>Enter whether you want to use independent (0) or dependent (1) gains. Displays one of the following:</p> <p>INDEPENDENT (0) – for independent gains DEPENDENT (1) – for dependent gains</p> <p>Use dependent gains when you want to use standard loop tuning methods. Use independent gains when you want the three gain constants (P, I, and D) to operate independently.</p>
Derivative of	.DO=0 .DO=1	<p>Enter whether you want the derivative of the PV (0) or the error (1). Displays one of the following:</p> <p>PV (0) – for PV derivative ERROR (1) – for error derivative</p> <p>Select the PV derivative for more stable control when you do not change the setpoint often. Select the error derivative for fast responses to setpoint changes when the algorithm can tolerate overshoots.</p>
Control Action	.CA=0 .CA=1	<p>Enter whether you want reverse (0) or direct acting (1). Displays one of the following:</p> <p>REVERSE (0) – for reverse acting (E = SP-PV) DIRECT (1) – for direct acting (E = PV-SP)</p>
PV Tracking	.PVT=0 .PVT=1	<p>Enter whether you do not (0) or do (1) want PV tracking. Displays one of the following:</p> <p>NO (0) – for no tracking YES (1) – for PV tracking</p> <p>Select no tracking if the algorithm can tolerate a bump when switching from manual to automatic control. Select PV tracking if you want the setpoint to track the PV in manual control for bumpless transfer to automatic control.</p>
Update Time	.UPD	<p>Enter an update time (greater than or equal to .01 seconds) at 1/5 to 1/10 the natural period of the load (load time constant). The load time constant should be greater than:</p> <p style="text-align: center;">3ms(algorithm) + block transfer time (ms)</p> <p>Periodically enable the PID instruction at a constant interval equal to the update time. When the program scan time is close to the required update time, use an STI to ensure a constant update interval. When the program scan is several times faster than the required update time, use a timer.</p> <p>Attention: If you omit an update time or enter a negative update time, a major fault occurs the first time the processor runs the PID instruction.</p>

(Continued)

Parameter:	Address Mnemonic:	Description:
Cascade Loop	.CL=0 .CL=1	Enter whether this loop is not (0) or is (1) used in a cascade of loops. Displays one of the following: NO (0) – not used in a cascade YES (1) – used in a cascade
Cascade Type	.CT=0 .CT=1	If this loop is part of a cascade of loops, enter whether this loop is the master (1) or a slave (0). Displays one of the following: SLAVE (0) – for a slave loop MASTER (1) – for a master loop
Master to this Slave	.ADDR	If this loop is a slave loop in a cascade, enter the control block address of the master. Tieback is ignored in the master loop of a cascade. When you change cascaded loops to manual control, the slave forces the master into manual control. When PV tracking is enabled, the order of events is: Slave.SP > Master.TIE > Master.OUT > Slave.SP When you return to automatic control, change the slave first, then the master.
Engineering Unit Max	.MAXS	Enter the floating-point value in engineering units that corresponds to the analog module's full scale output. Valid range is $-3.4 E^{+38}$ to $+3.4 E^{+38}$. Attention: Do not change this value during operation because a processor fault might occur.
Engineering Unit Min	.MINS	Enter the floating-point value in engineering units that corresponds to the analog module's zero output. Valid range is $-3.4 E^{+38}$ to $+3.4 E^{+38}$ (post-scaled number). Attention: Do not change the maximum scaled value during operation because a processor fault might occur.
Input Range Max	.MAXI	Enter the floating-point number ($-3.4 E^{+38}$ to $+3.4 E^{+38}$) that is the unscaled maximum value available from the analog module. For example, use 4095 for a module whose range is 0-4095.
Input Range Min	.MINI	Enter the floating-point number ($-3.4 E^{+38}$ to $+3.4 E^{+38}$) that is the minimum unscaled value available from the analog module. For example, use 0 for a module whose range is 0-4095.
Output Limit High %	.MAXO	Enter a percent (0-100) above which the algorithm clamps the output.
Output Limit Low %	.MINO	Enter a percent (0-100) below which the algorithm clamps the output.
PV Alarm High	.PVH	Enter a floating-point number ($-3.4 E^{+38}$ to $+3.4 E^{+38}$) that represents the highest PV value that the system can tolerate.
PV Alarm Low	.PVL	Enter a floating-point number ($-3.4 E^{+38}$ to $+3.4 E^{+38}$) that represents the lowest PV value that the system can tolerate.
PV Alarm Deadband	.PVDB	Enter a floating-point number ($0-3.4 E^{+38}$) that is sufficient to minimize nuisance alarms. This is a one-sided deadband. The alarm bit (.PVH or .PVL) is not set until the PV crosses the deadband and reaches the alarm limit (DB zero point). The alarm bit remains set until the PV passes back through and exits from the deadband.

(Continued)

Parameter:	Address Mnemonic:	Description:
Deviation Alarm (+)	.DVP	Enter a floating-point number ($0-3.4 E^{+38}$) that specifies the greatest error deviation above the setpoint that the system can tolerate.
Deviation Alarm (-)	.DVN	Enter a floating-point number ($-3.4 E^{+38}-0$) that specifies the greatest error deviation below the setpoint that the system can tolerate.
Deviation Alarm Deadband	.DVDB	Enter a floating-point number ($0-3.4 E^{+38}$) that is sufficient to minimize nuisance alarms. This is a one-sided deadband. The alarm bit (.DVP or .DVN) is not set until the error crosses the deadband and reaches the alarm limit (DB zero point). The alarm bit remains set until the error passes back through and exits from the deadband.
No Zero Crossing	.NOZC=0 .NOZC=1	Enter whether to use (1) or inhibit (0) the no zero crossing feature: 0 – no zero crossing disabled 1 – no zero crossing enabled
No Back Calculation	.NOBC=0 .NOBC=1	Enter whether to use (1) or inhibit (0) the no back calculation feature: 0 – no back calculation disabled 1 – no back calculation enabled
No Derivative Filter	.NDF=0 .NDF=1	Enter whether to use (1) or inhibit (0) the filter in the derivative calculation. 0 – no filter used in derivative calculation 1 – filter used in derivative calculation

Using Control Block Values

Words 0 and 1 of the control block contains status and control bits. Table 14.F shows the values stored in each word of the control block.

Table 14.F
PID Control Block

Word:	Contains:	Range:
0	Control/Status Bits Bit 15 Enabled (EN) Bit 11 No back calculation (0=disabled, 1=enabled) Bit 10 No zero crossing (0=disabled, 1=enabled) Bit 9 Cascade selection (master, slave) Bit 8 Cascade loop (0=no, 1=yes) Bit 7 Process variable tracking (0=no, 1=yes) Bit 6 Derivative action (0=PV, 1=error) Bit 5 No derivative filter (0=disabled, 1=enabled) Bit 4 Set output (0=no, 1=yes) Bit 2 Control action (0=SP-PV, 1=PV-SP) Bit 1 Mode (0=automatic, 1>manual) Bit 0 Equation (0=independent, 1=ISA)	

(Continued)

Word:	Contains:	Range:
1	Status Bits Bit 12 PID initialized (0=no, 1=yes) Bit 11 Set point out of range Bit 10 Output alarm, lower limit Bit 9 Output alarm, upper limit Bit 8 DB, set when error is in DB Bit 3 Error is alarmed low Bit 2 Error is alarmed high Bit 1 Process variable (PV) is alarmed low Bit 0 Process variable (PV) is alarmed high Note: During prescan, bit 12 is cleared.	
2, 3	Setpoint	$-3.4 E^{+38}$ to $+3.4 E^{+38}$
4, 5	Independent: Proportional gain (unitless)	0 to $+3.4 E^{+38}$
	ISA: Controller gain (unitless)	0 to $+3.4 E^{+38}$
6, 7	Independent: Integral gain (1/sec)	0 to $+3.4 E^{+38}$
	ISA: Reset term (minutes per repeat)	0 to $+3.4 E^{+38}$
8, 9	Independent: Derivative gain (seconds)	0 to $+3.4 E^{+38}$
	ISA: Rate term (minutes)	0 to $+3.4 E^{+38}$
10, 11	Feedforward or bias	-100 to +100%
12, 13	Maximum scaling	$-3.4 E^{+38}$ to $+3.4 E^{+38}$
14, 15	Minimum scaling	$-3.4 E^{+38}$ to $+3.4 E^{+38}$
16, 17	Dead band	0 to $+3.4 E^{+38}$
18, 19	Set output	0-100%
20, 21	Maximum output limit (% of output)	0-100%
22, 23	Minimum output limit (% of output)	0-100%
24, 25	Loop update time (seconds)	
26, 27	Scaled PV value (displayed)	
28, 29	Scaled error value (displayed)	
30, 31	Output (% of 4095)	0-100%
32, 33	Process variable high alarm value	$-3.4 E^{+38}$ to $+3.4 E^{+38}$
34, 35	Process variable low alarm value	$-3.4 E^{+38}$ to $+3.4 E^{+38}$
36, 37	Error high alarm value	0 to $+3.4 E^{+38}$
38, 39	Error low alarm value	$-3.4 E^{+38}$ to 0
40, 41	Process variable alarm deadband	0 to $+3.4 E^{+38}$
42, 43	Error alarm deadband	0 to $+3.4 E^{+38}$

(Continued)

Word:	Contains:	Range:
44, 45	Maximum input value	$-3.4 E^{+38}$ to $+3.4 E^{+38}$
46, 47	Minimum input value	$-3.4 E^{+38}$ to $+3.4 E^{+38}$
48, 49	Tieback value for manual control (0-4095)	0-100%
51	Master PID file number	0-999; 0-9999 for Enhanced PLC-5 processors only
52	Master PID element number	0-999; 0-9999 for Enhanced PLC-5 processors only
54-80	internal storage; do not use	

Programming Considerations

When you program a PID instruction, do not change the following values when the processor is in Run mode:

- choice of ISA or independent gains equation because the PID gains constants are not directly interchangeable
- scaling values S_{\min} and S_{\max} because a change could place the setpoint out of range and could change the dead-band range
- choice of derivative action based on change in PV or change in error because internal values will change

Run Time Errors

If the setpoint (SP) is out of range ($SP < S_{\min}$ or $SP > S_{\max}$), the processor produces a run time error when it executes the instruction.

If you change SP, S_{\min} , or S_{\max} to create the latter condition, the PID instruction first tries to use the previously valid setpoint, continues PID control, and sets the setpoint out of range error bit. If the instruction finds no previously valid setpoint, it produces a run time error.

If you enter negative values for K_p , K_I , K_D , K_C , T_I , or T_D , the PID instruction substitutes zero for the negative value. This inhibits that term in the equation without producing a run time error.

Transferring Data to the PID Instruction

Use block transfer instructions to transfer data between analog I/O modules and the PID instruction. Use a BTR instruction for input values (process variable and tieback); use a BTW instruction for the control output.

Make each block transfer file address (data file entry) the same address in the PID for the process variable, tieback, and control output, respectively.

Not all Allen-Bradley analog input modules enter data in the same format. You must determine where to store channel data. For example, temperature sensing modules (such as the 1771-IR and 1771-IXE) place status words in front of the words that contain channel data. For information about where an analog module stores channel data, see the documentation for the module.

Loop Considerations

The number of PID loops, loop update time, and location of 12-bit analog input modules are important considerations for using the PID instruction.

Number of PID Loops

The number of PID loops that the processor can handle depends on the update time required by the loops. The longer the update time and the less sophisticated the loop control, the more loops the processor can control.

The sum of the worst-case block transfer time associated with the analog inputs plus the time required for one program scan should be less than the update time required by the loops.

Loop Update Time

The PID instruction calculates a new control output whenever its rung changes from false to true when using an integer data file for the control block. A PID instruction with a PD control block will execute every scan in which the rung is true. You can use a one shot instruction to force the PID instruction with a PD control block to only execute on a false to true transition. See the examples at the end of this chapter. For the instruction to operate predictably, the update time must be equal to the rate at which the PID rung changes between false and true. Deviation in toggle rate from the update time substantially degrades the accuracy of PID calculations.

You should program fast response loops (update times less than 100 ms) in the selectable timed interrupt (STI) along with the corresponding block transfer instructions. Unlatch the PID enable bit to force execution every STI scan (if you are using a PD data file for the control block you do not have to unlatch the enable bit). You must place corresponding analog I/O modules in the local chassis when you see this configuration.

Program slower response loops (update times of greater than 100 ms) in the main ladder program and use timers or real-time sampling to control the update time.

Descaling Inputs

The PID instruction must use unscaled (0-4095) data from analog input modules. The analog input modules you can use may have either scaled or unscaled ranges. When possible, select the unscaled range of 0-4095.

However, some modules such as the 1771-IR and 1771-IXE temperature sensing modules, cannot generate data in an unscaled range. For these modules, you must program arithmetic logic to convert the scaled output to the unscaled range for the PID instruction. If you are using a PD data file for the control block, the processor performs this descaling internally (see the descriptions of .MAXI and .MINI in the PID configuration characteristics, page 14-22).

Use this equation to convert scaled outputs:

$$M_2 = (M_1 - S_{\min 1}) \frac{4095}{(S_{\max 1} - S_{\min 1})}$$

Variable	Description
M ₂	calculated output
M ₁	measured value from the module in scaled units
S _{max1}	scaled maximum value from the module
S _{min1}	scaled minimum value from the module
S _{max1} - S _{min1}	scaled range from the module

For example, the reading from a 1771-IXE module for type J thermocouple is 170°. To convert this to an unscaled value, use these values:

$$M_2 = [170 - (-200)] \frac{4095}{[1200 - (-200)]}$$

$$M_2 = 1082 \quad \text{unscaled}$$

If you are sure that the temperature of your process will always remain within a specific range, you can set the limits for S_{min1} and S_{max1} instead of the minimum and maximum values for the thermocouple module. This technique improves the resolution of the process variable.



ATTENTION: If you set the limits instead of using the lower and upper temperature limits of the thermocouple or RTD module, you must keep the process within the limits you specify. Failure to keep the process within the limits could cause unpredictable operation, damage to equipment, or injury to personnel.

Figure 14.3 shows the ladder logic you need to add to your PID program. Table 14.G lists the variables in this example.

Figure 14.3
Descaling PID Values Example

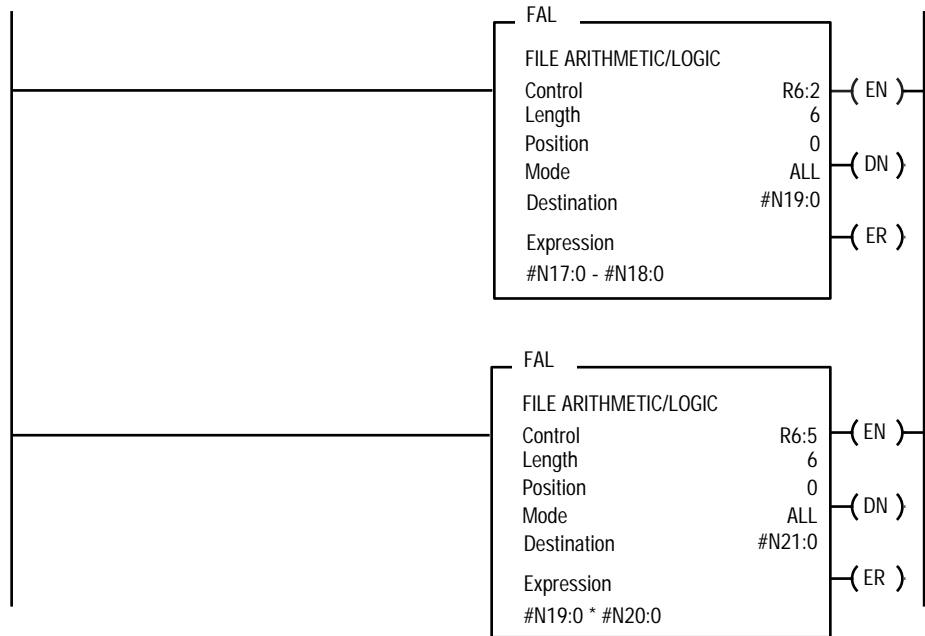


Table 14.G
Variables for the Descaling PID Values Example

Variable	Description
S_{max}	maximum scaling value
S_{min}	minimum scaling value
$K = \frac{4095}{S_{max} - S_{min}}$	constant for each channel
#N17:0	contains M_1 values for each channel
#N18:0	contains S_{min} constants for each channel
#N19:0	contains the result of $M_1 - S_{min}$ for each channel
#N20:0	location where you store K for each channel
#N21:0	contains the resulting unscaled value for each channel

PID Examples

The following examples assume that the channel data is stored starting at the beginning (first word) of the block transfer file.

Integer Block (N) Examples

Main Program File

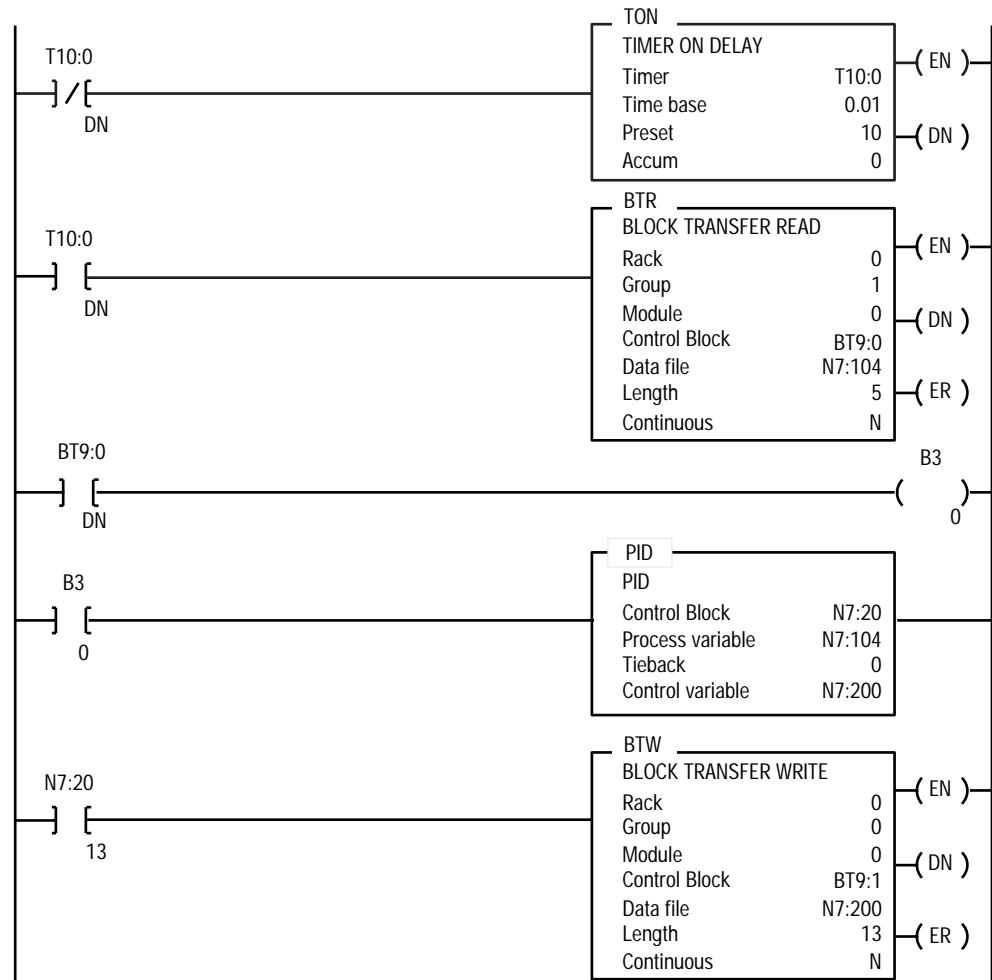
When you place the PID instruction in the main program file, control the sample time with a timer, where the PID Loop Update Time = timer preset.

Timer based execution uses a free-running timer for event coordination. When the timer's accumulated value reaches its preset value, it triggers the loop update sequence. The timer immediately resets and restarts to maintain a consistent update interval. Use timer based execution in "slower" loop applications or in applications with relatively few loops. See Figure 14.4 for programming example.

The accuracy of the timer depends on the time base and the total scan time of the processor. Always choose the 0.01 second time base for this PID application. Duplicate the timer instruction elsewhere in the program if the processor scan time (local I/O scan plus program scan) is greater than 2.5 seconds.

Because block transfers in the local chassis occur asynchronously during main program scan, you need a storage bit to ensure that the state of the PID input condition remains constant during the entire program scan. Condition all PID instructions using this storage bit.

Figure 14.4
Example PID Programming Conditioned by a Timer in the Main Program



STI Program File

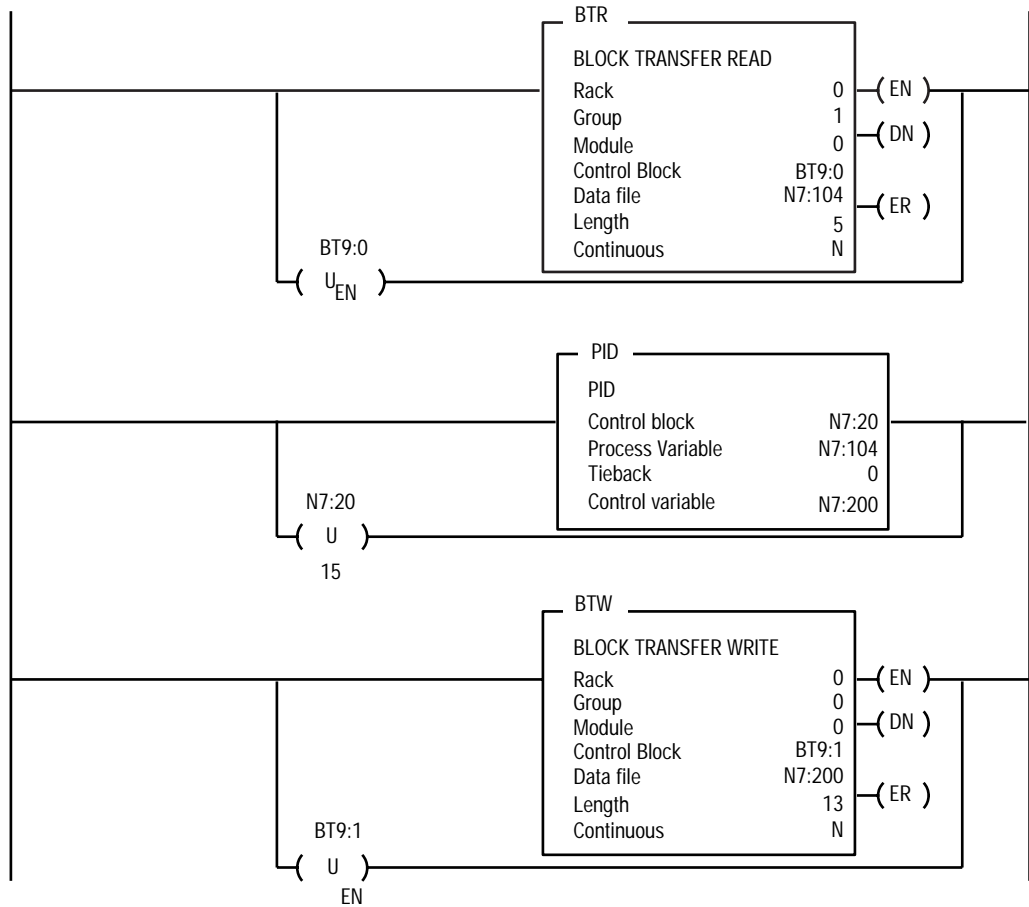
When you place the PID instruction in a selectable timed interrupt file (STI), the STI controls the loop update (sampling) time where the PID Loop Update Time = STI interval.

In the STI, a separate program file contains all of the necessary logic to accomplish the loop update. The PLC-5 processor is configured with an STI to execute that file at the user's update interval. STI loop coordination is desirable with "faster" loops or when more loop processing is required at the specified update interval. See Figure 14.5 for programming examples.

The PID instruction operates on the most recent data when block transfer instructions are included in the STI file. You must place block transfer modules in the local chassis for this PID application. Unlatching the PID and BT enable bits forces the processor to run the PID and block transfer instructions every time the STI is enabled.

Important: The program scan waits for block transfer instructions in the STI file to complete their transfers.

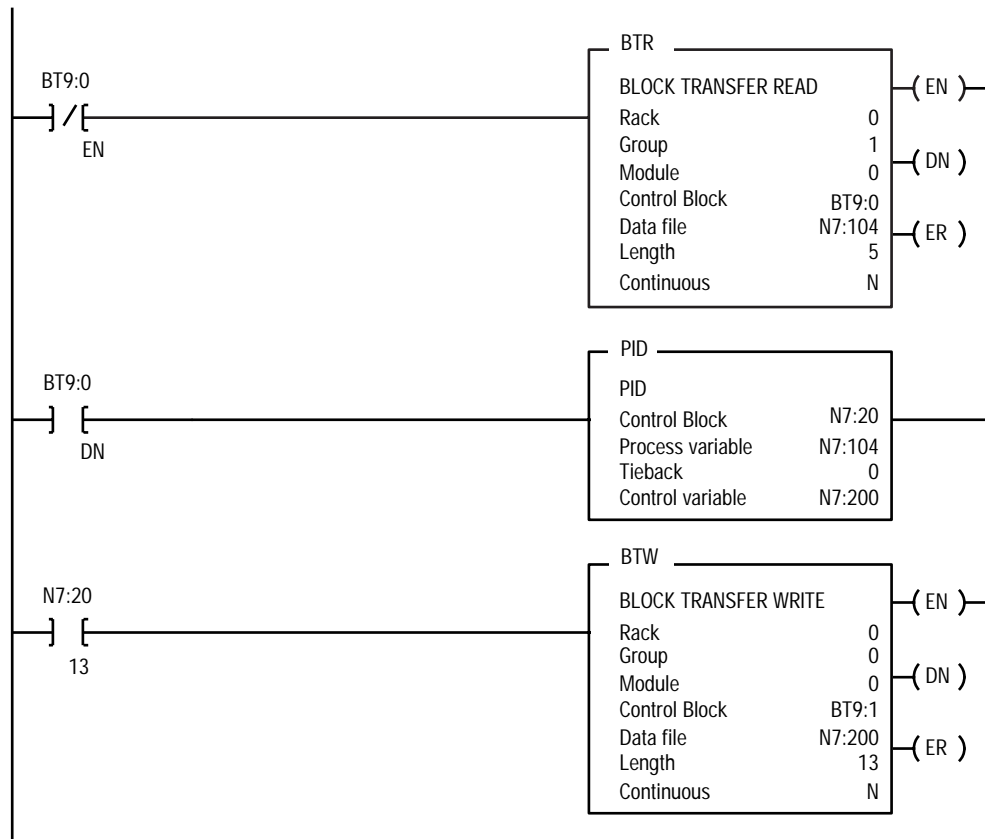
Figure 14.5
Example PID Programming in an STI File



RTS Program File

With the Real Time Sample Based (RTS), the PID instruction's execution is triggered by the availability of new analog data from an analog input source configured for real time sampling. Since the RTS configuration of an analog module will not initiate or allow a BTR until new data is available, the PID instruction's rung can be conditioned by the BTR's done bit. This assures that the PID instruction is executed only when new analog data is available at the RTS interval. See Figure 14.6 for programming examples where the PID Loop Update Time = RTS interval.

Figure 14.6
Example PID Programming in an RTS File



PD Block Examples

Main Program File

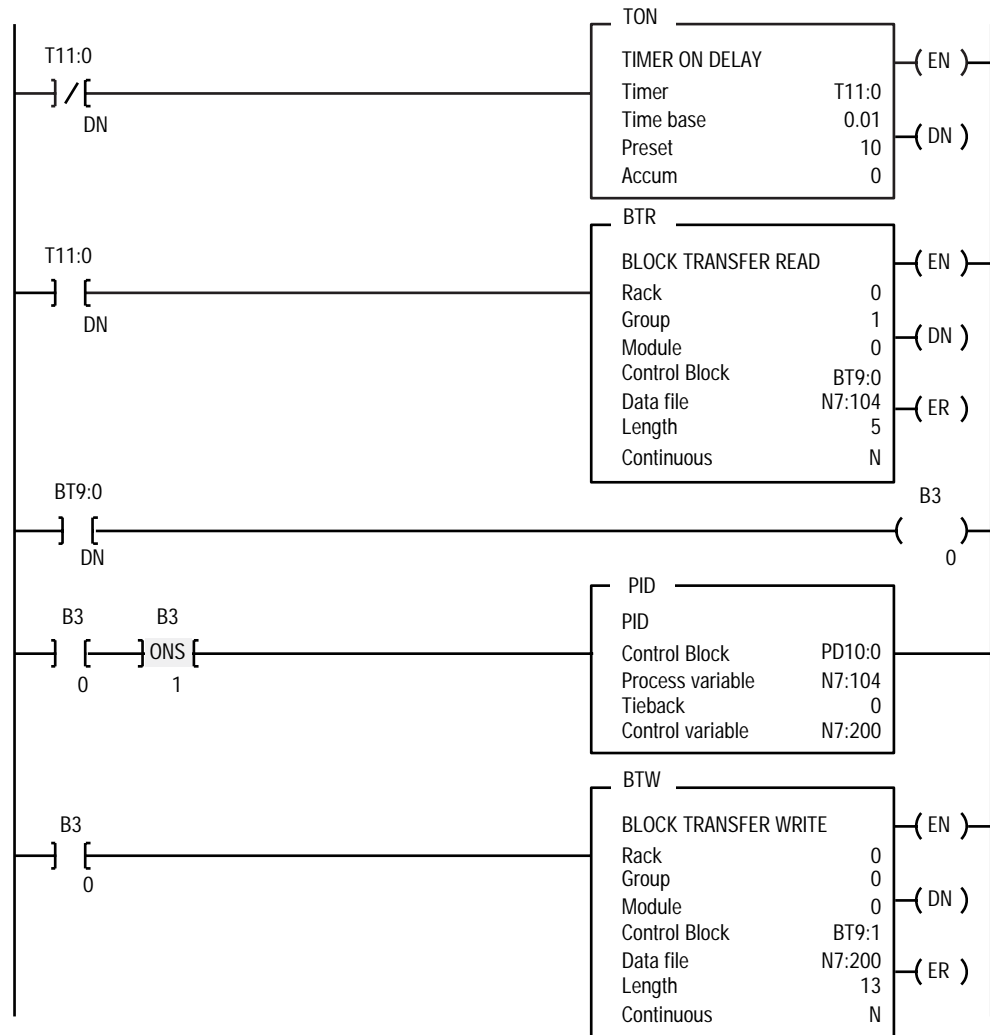
When you place the PID instruction in the main program file, control the sample time with a timer, where the PID Loop Update Time = timer preset.

Timer based execution uses a free-running timer for event coordination. When the timer's accumulated value reaches its preset value, it triggers the loop update sequence. The timer immediately resets and restarts to maintain a consistent update interval. Use timer based execution in "slower" loop applications or in applications with relatively few loops. See Figure 14.7 for programming example.

The accuracy of the timer depends on the time base and the total scan time of the processor. Always choose the 0.01 second time base for this PID application. Duplicate the timer instruction elsewhere in the program if the processor scan time (local I/O scan plus program scan) is greater than 2.5 seconds.

Because block transfers in the local chassis occur asynchronously during main program scan, you need a storage bit to ensure that the state of the PID input condition remains constant during the entire program scan. Condition all PID instructions using this storage bit.

Figure 14.7
Example PID Programming Conditioned by a Timer in the Main Program



STI Program File

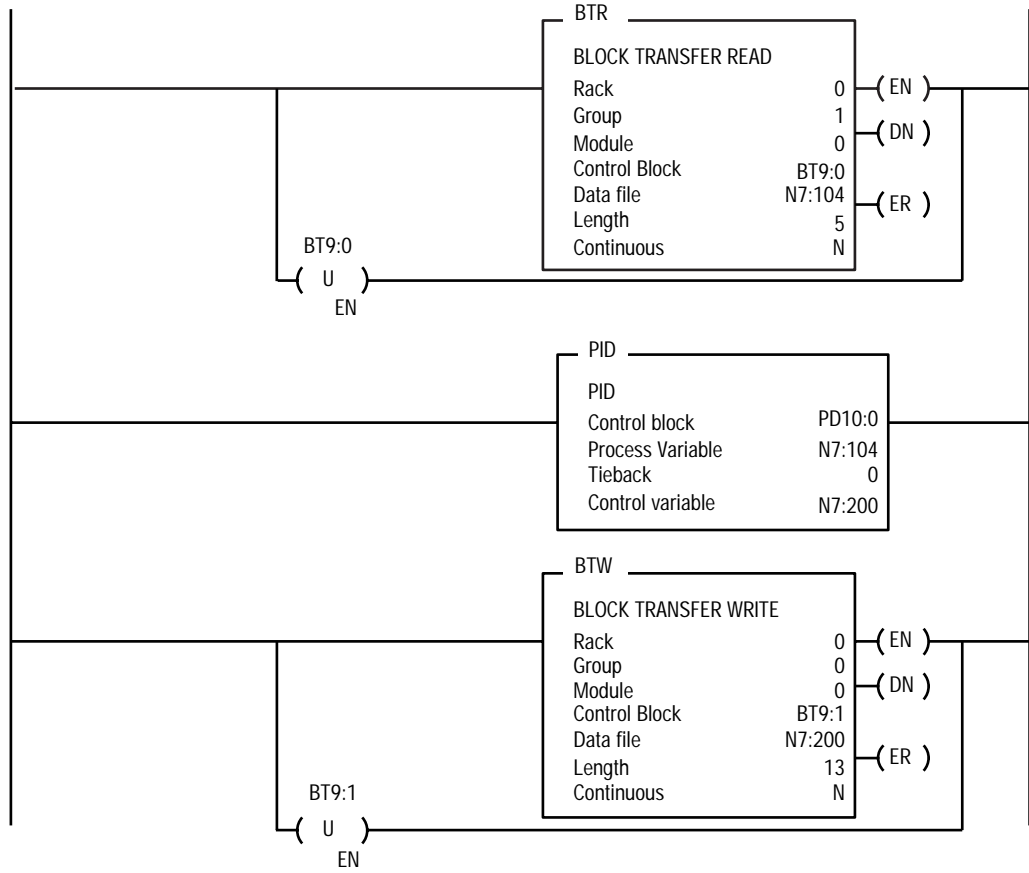
When you place the PID instruction in a selectable timed interrupt file (STI), the STI controls the loop update (sampling) time where the PID Loop Update Time = STI interval.

In the STI, a separate program file contains all of the necessary logic to accomplish the loop update. The PLC-5 processor is configured with an STI to execute that file at the user’s update interval. STI loop coordination is desirable with “faster” loops or when more loop processing is required at the specified update interval. See Figure 14.8 for programming examples.

The PID instruction operates on the most recent data when block transfer instructions are included in the STI file. You must place block transfer modules in the local chassis for this PID application. Unlatching the PID and BT enable bits forces the processor to run the PID and block transfer instructions every time the STI is enabled.

Important: The program scan waits for block transfer instructions in the STI file to complete their transfers.

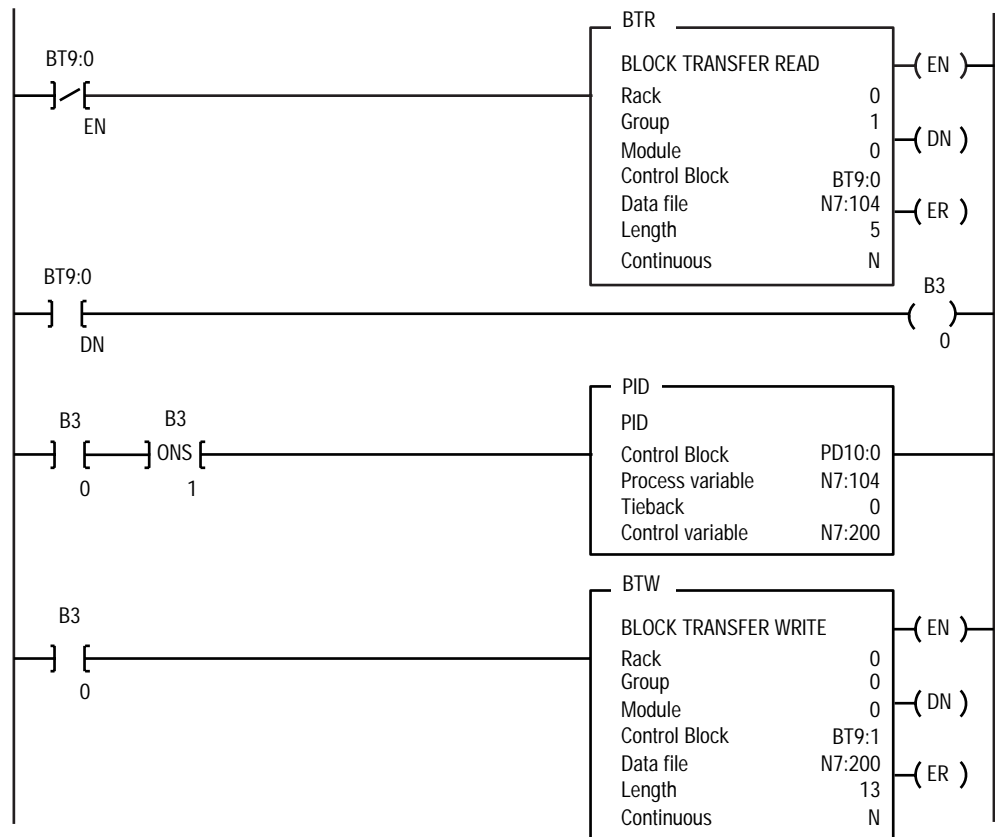
Figure 14.8
Example PID Programming in an STI File



RTS Program File

With the Real Time Sample Based (RTS), the PID instruction's execution is triggered by the availability of new analog data from an analog input source configured for real time sampling. Since the RTS configuration of an analog module will not initiate or allow a BTR until new data is available, the PID instruction's rung can be conditioned by the BTR's done bit. This assures that the PID instruction is executed only when new analog data is available at the RTS interval. See Figure 14.9 for programming examples where the PID Loop Update Time = RTS interval.

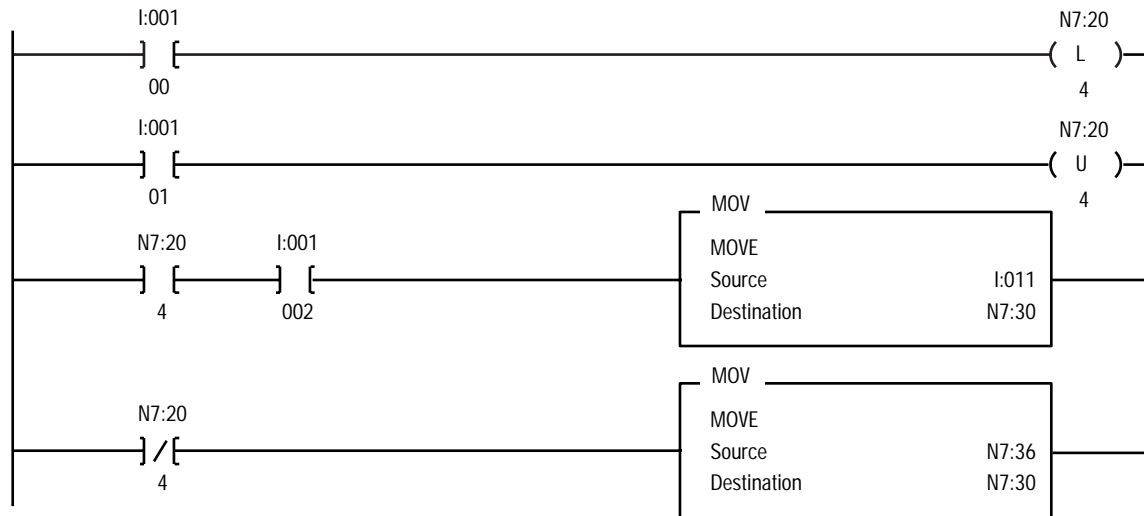
Figure 14.9
Example PID Programming in an RTS File



Ladder Logic Simulation of a Manual Control Station

When you program the simulation of a manual control station, make sure that a hardware manual control station is not connected when the program is enabled. Add the rungs in Figure 14.10 to the PID program in Figure 14.4, Figure 14.5, Figure 14.7, or Figure 14.8.

Figure 14.10
Example Program for Simulating a Manual Control Station



The last rung in the above example is for output tracking for bumpless transfer from automatic to manual mode.

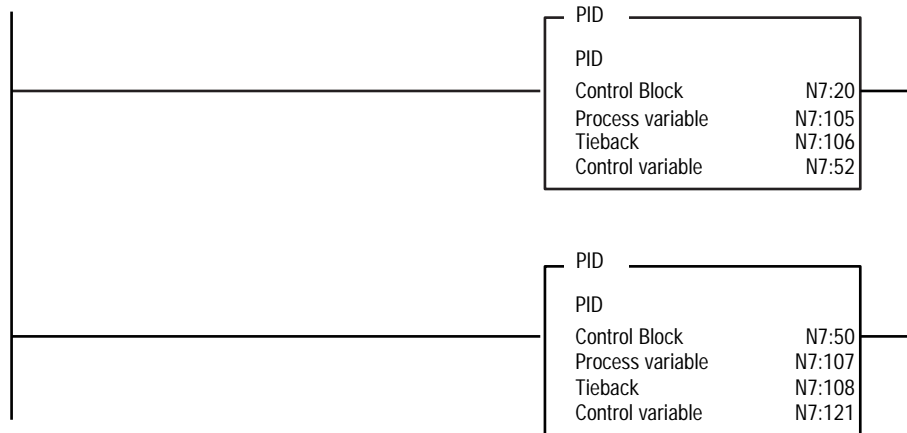
Address:	Description:
I:001/00	Manual pushbutton switch
I:001/01	Automatic pushbutton switch
I:001/02	Enter pushbutton switch
I:011	Manual output value
N7:20/4	PID set output bit
N7:30	PID set output value
N7:36	Current control output

Cascading Loops

You can cascade two loops by assigning the control output of the outer loop to the setpoint of the inner loop. The setpoint of the inner loop is the third word (word 2) of the integer control block. If the control block of the inner loop is N7:50, address the outer loop control output at N7:52. Replace the PID rungs in Figure 14.14 or Figure 14.5 with those in Figure 14.11.

You must not scale the setpoint of the inner loop. Set the scaling bit (word 0, bit 5) to 1 to inhibit setpoint scaling.

Figure 14.11
Cascaded Loops



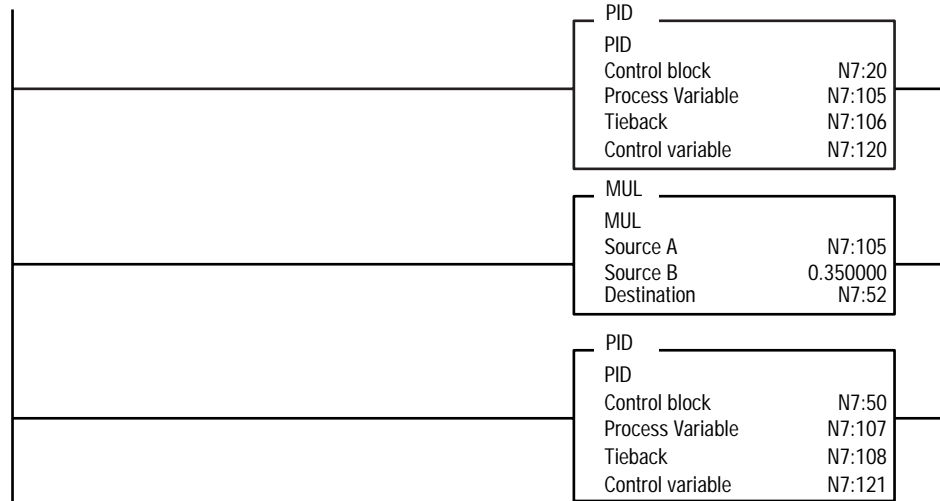
Ratio Control

You can maintain two values in a ratio by using a MUL instruction. Three parameters are involved:

- the wild or uncontrolled value
- the controlled value
- the ratio between these two values

Enter the address of the controlled value as the Destination. Enter the address of the wild or uncontrolled value as Source A. Enter either the address of the ratio value or a program constant for the ratio as Source B. For example, add the rungs in Figure 14.12 to the PID program in Figure 14.4 or Figure 14.5.

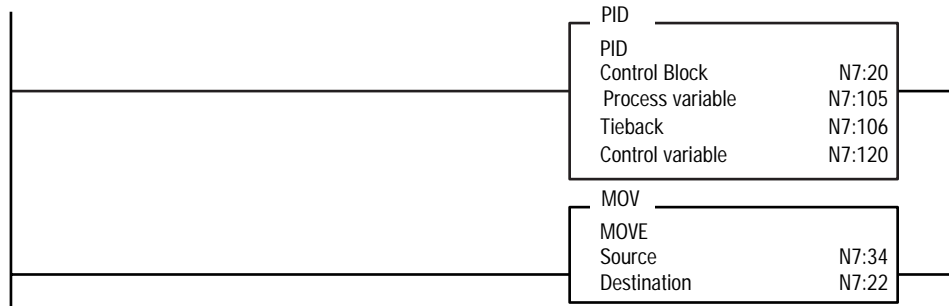
Figure 14.12
Ratio Control with a PID Instruction



Process Variable Tracking

When in manual control, your program can force the setpoint to be equal to the process variable (PV) by moving the PV into the setpoint word (word 2 of the integer control block) to achieve a smooth manual-to-automatic transfer. If the setpoint is scaled, move the scaled PV from the PID control block directly into the setpoint word. If the setpoint is not scaled, move the unscaled value from the PV address in the PID instruction to the setpoint. For an example, add the rungs in Figure 14.13 to the PID program in Figure 14.4 or Figure 14.5.

Figure 14.13
Process Variable Tracking



PID Theory

Figure 14.14 and Figure 14.15 show the PLC-5 PID Integer and PD Block process flow. Figure 14.16 and Figure 14.17 show the PD Block master-slave relationship.

Figure 14.14
PLC-5 PID (Integer Block)

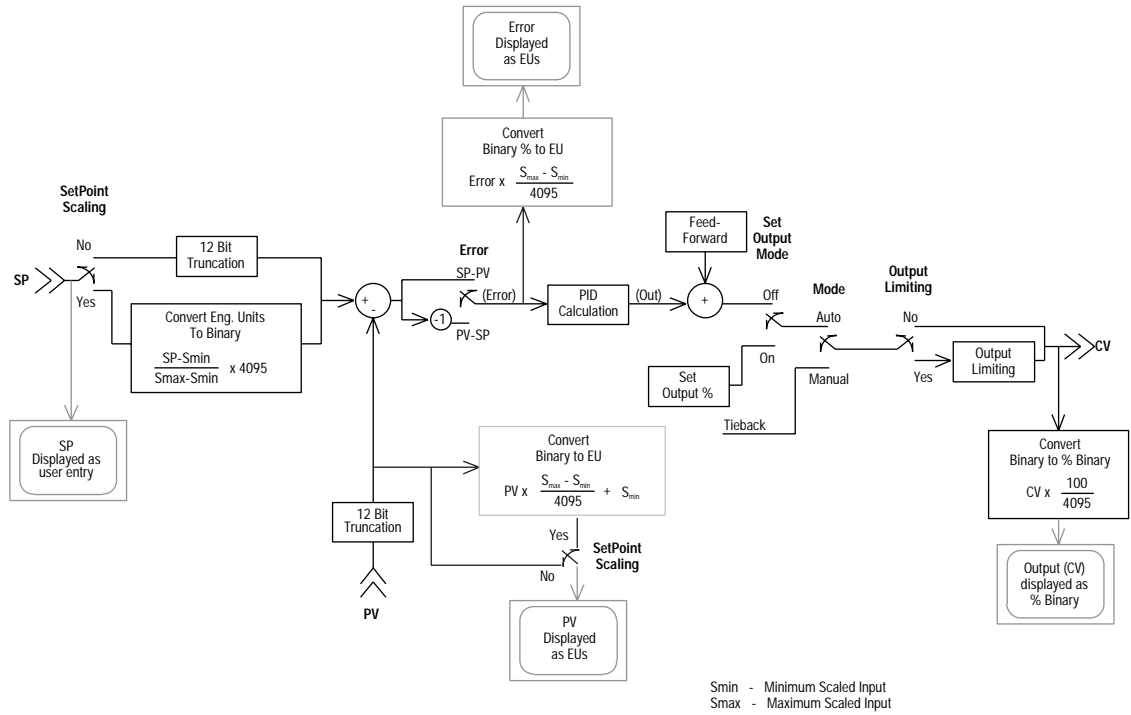


Figure 14.15
PLC-5 PID (PD Block)

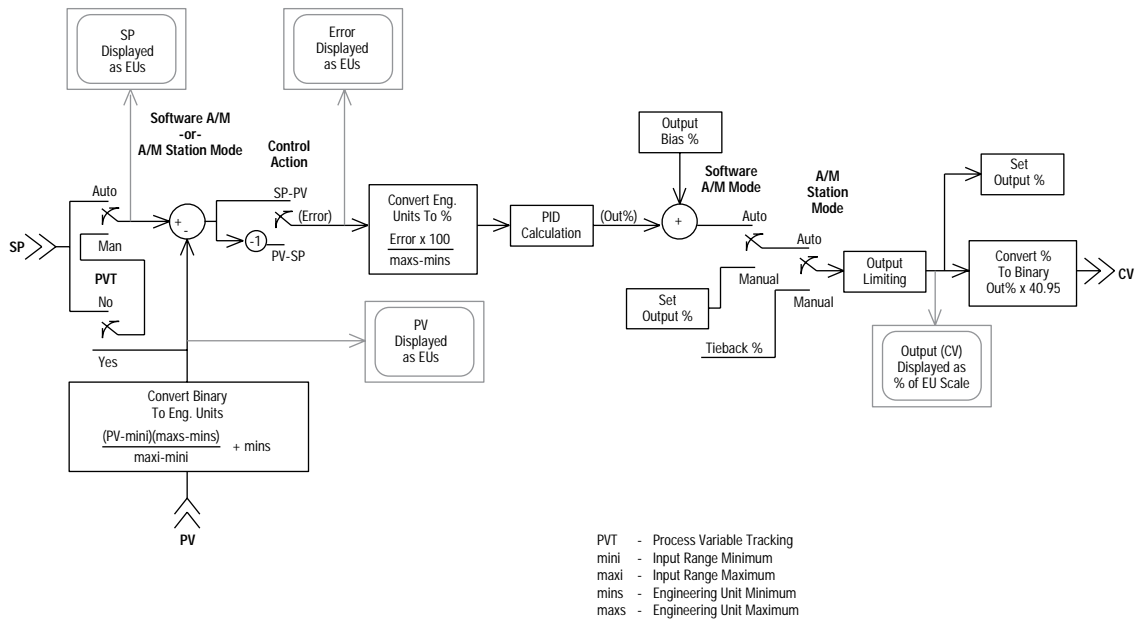


Figure 14.16
 PLC-5 PID (PD Block) as Master/Slave Loops

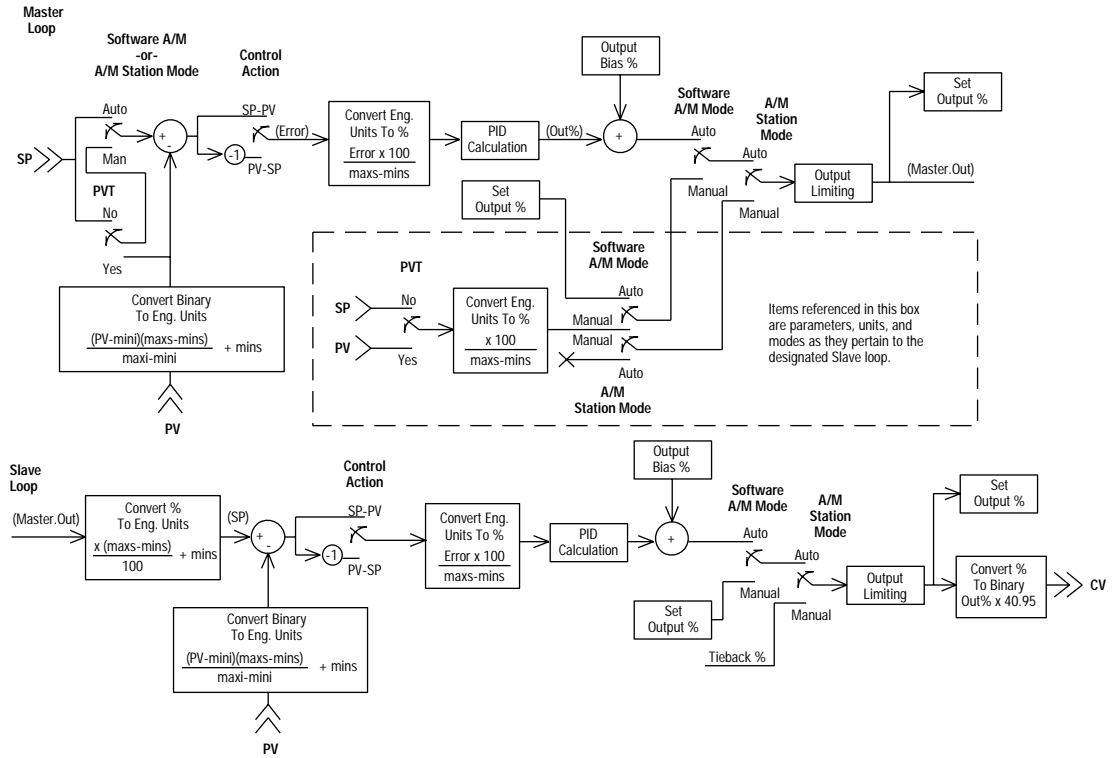
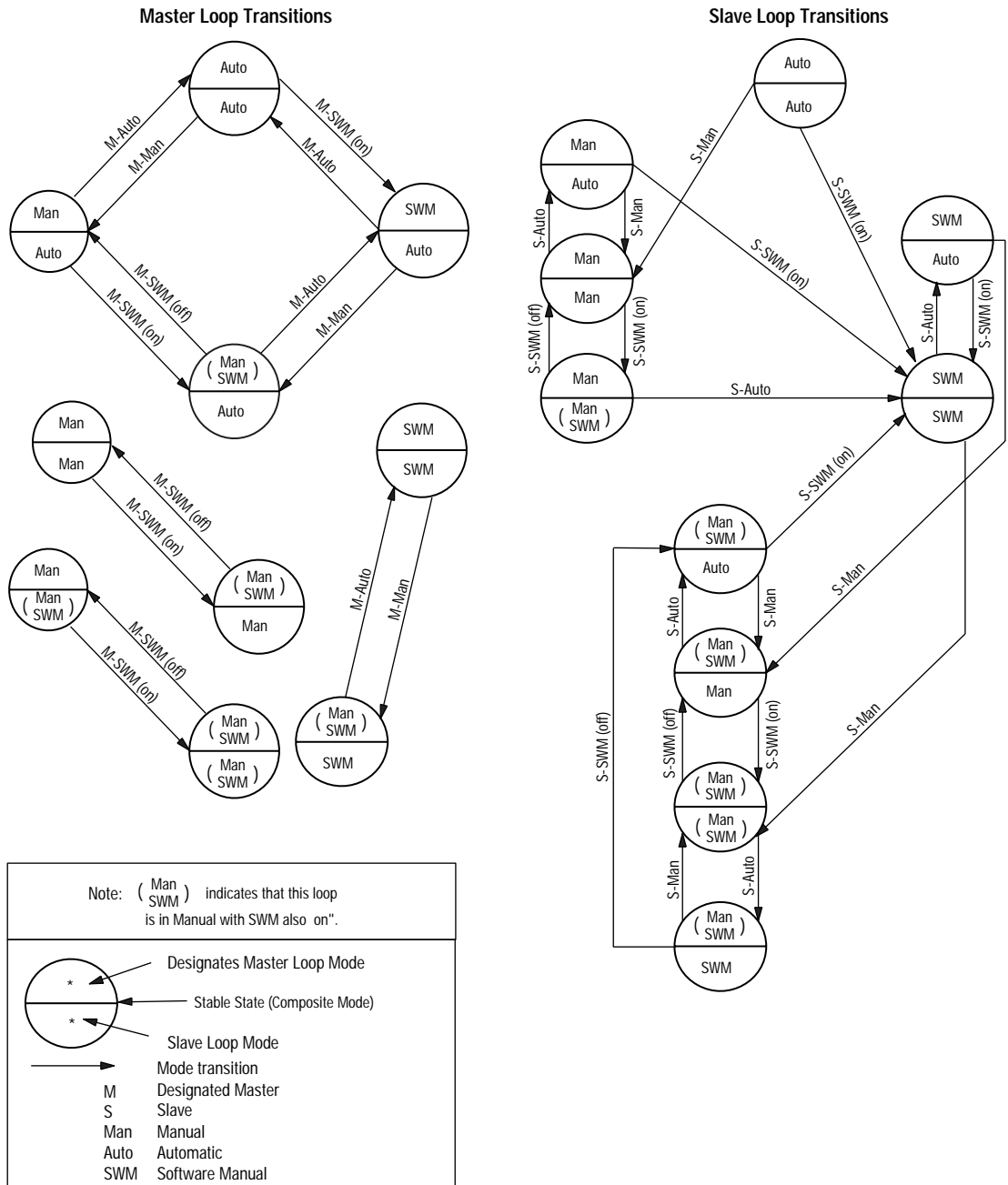


Figure 14.17
PD Block Master/Slave Interlocking State Transitions



Block-Transfer Instructions BTR and BTW ControlNet I/O Transfer Instruction CIO

Using Block Transfer and ControlNet I/O Transfer Instructions

Block transfer instructions let you transfer words to or from a block transfer module; ControlNet I/O transfer instructions let you perform unscheduled transfers to I/O modules on a ControlNet™ network. Table 15.A lists the available block transfer and ControlNet I/O transfer instructions.

Table 15.A
Available Block Transfer and ControlNet I/O Transfer Instructions

If You Want to:	Use this Instruction:	Found on Page:
Transfer words to a block transfer module	BTW	15-3
Transfer words from a block transfer module	BTR	15-3
Perform unscheduled transfers to I/O modules on a ControlNet network	CIO	15-22

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Using Block Transfer Instructions

With block-transfer instructions, you can transfer up to 64 words at a time to or from a block transfer module in a local or remote I/O chassis. You can also transfer up to 64 words at a time between a supervisory processor (scanner mode) and a processor configured for adapter mode.

The Enhanced PLC-5 processors have configurable communication channels; choose between remote I/O scanner, remote I/O adapter, or DH+. Ladder block transfer instructions are not necessary when using Enhanced PLC-5 processors in adapter mode.

Table 15.B describes how to block transfer data to a local or remote rack when the processor is configured for scanner mode. Figure 15.1 illustrates how the transfer occurs.

Table 15.B
Block Transfer Instructions for Local or Remote Racks in Scanner Mode

If You Want to Transfer Data:	Use:
To the BT I/O module	BTW (block-transfer write)
From the BT I/O module	BTR (block-transfer read)

Figure 15.1
Block Transfer Operation in Scanner Mode

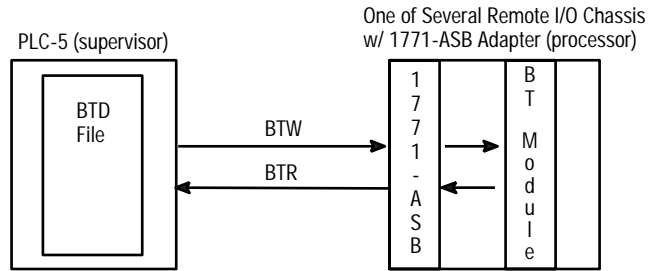
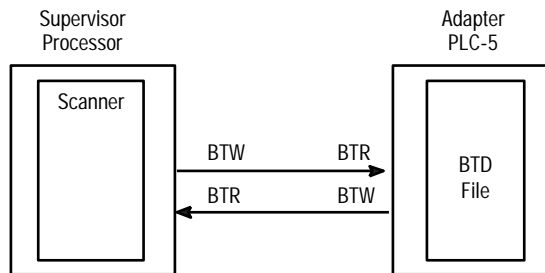


Table 15.C describes how to block transfer data when the processor is configured for adapter mode. Figure 15.2 illustrates how the transfer occurs.

Table 15.C
Block Transfer Instructions for Adapter Mode

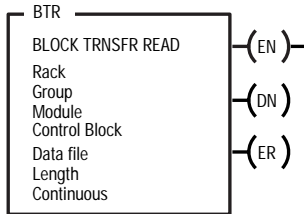
If You Want to Transfer Data:	Use:
From the supervisory processor	BTR (block-transfer read)
To the supervisory processor	BTW (block-transfer write)

Figure 15.2
Block Transfer Operation in Adapter Mode



Both processors simultaneously execute the opposite block transfer instruction.

Block-Transfer Read (BTR) and Block-Transfer Write (BTW)



Description:

When the rung goes true, the BTW instruction tells the processor to write data stored in the data file to the specified rack/group/module address; the BTR instruction tells the processor to read data from the rack/group/module address and store it in the data file.

Block-Transfer Request Queue

When a false-to-true rung transition enables a BTW or BTR instruction, the transfer request is queued:

For this Processor:	The Queue Holds Up to:
Classic PLC-5	17 block transfer requests per logical rack
PLC-5/11, 5/20, -5/30	64 block transfer requests to remote racks (maximum 64 per channel pair – 1A/1B); no limit for requests to local racks
PLC-5/40, -5/60, -5/80	128 block transfer requests to remote racks (maximum 64 per channel pair – 1A/1B, 2A/2B); no limit for requests to local racks

The processor runs each block transfer request in the order it is requested. When the processor changes to Program mode, any block transfers are cancelled.

For Classic PLC-5 processors, each rack number has a block transfer queue with a corresponding queue-full bit. Table 15.D lists the queue-full bits. Once these bits are set, your ladder program must clear them. Your program should continually monitor these queue-full bits, found in the status file, word 7, bits 08-15. (Enhanced PLC-5 processors can have unlimited block transfers in local racks, so there are no queue-full bits.)

Table 15.D
Queue-Full Bits for Block Transfer Requests (Word 7) –
Classic PLC-5 Processors

Bit	Description
S:7/8	Block-transfer queue for rack 0 is full
S:7/9	Block-transfer queue for rack 1 is full
S:7/10	Block-transfer queue for rack 2 is full
S:7/11	Block-transfer queue for rack 3 is full
S:7/12	Block-transfer queue for rack 4 is full
S:7/13	Block-transfer queue for rack 5 is full
S:7/14	Block-transfer queue for rack 6 is full
S:7/15	Block-transfer queue for rack 7 is full

The number of racks in your system depends on the processor you use.

A BTR or BTW instruction writes values into its control block address (a five-word integer file) when the instruction is entered. The processor uses these values to execute the transfer.

The Enhanced PLC-5 processors also have a block transfer file type (BT). You can still use existing programs with integer file types, but the new BT file type makes addressing easier. For example, if you need two control files, you can use BT10:0 and BT10:1; using integer files, you would have to use, for example, N7:0 and N7:5.

Entering Parameters

To program a BTW or BTR instruction, you must provide the processor with the following information that it stores in its control block:

- **Rack** is the I/O rack number (00-27 octal) of the I/O chassis in which you placed the target I/O module. Table 15.E lists the valid ranges for rack numbers.

Table 15.E
Valid Ranges for Rack Number in Block Transfer Instructions

Processor	Maximum Racks	Valid Range for Rack Numbers (octal)
PLC-5/10, -5/11, -5/12, -5/15, -5/20, -5/VME	4	00-03
PLC-5/25, -5/30	8	00-07
PLC-5/40, -5/40L	16	00-17
PLC-5/60, -5/60L, -5/80	24	00-27

- **Group** is the I/O group number (0-7) which specifies the position of the target I/O module in the I/O chassis.
- **Module** is the slot number (0-1) within the group. When using 2-slot addressing, the 0 slot is the low slot; the 1 slot is the high slot. You should use 0 for the module when using 1- or 1/2-slot addressing.
- **Control Block** is a six-word block transfer control file (BT) or a five-word integer file (N) that controls the instruction's operation. Enter this file address without the # symbol. This is not a control file (type R).

Important: You can use indirect addresses for the control block address in a BTR or BTW instruction.

Important: In a PLC-5/40, -5/60, or -5/80 processor, the block transfer data type (BT) must be used for rack addresses greater than 7.

The five-word integer (N) control file has the following structure:

	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
word 0	EN	ST	DN	ER	CO	EW	NR	TO	RW	rack			group		slot	
word 1	requested word count															
word 2	transmitted word count															
word 3	file-type number															
word 4	element number															

For information on the status bits in word 0, see page 15-8; for information on words 1 through 4, see page 15-10.

- **Data File** is the address of the input, output, status, integer (N), float, binary, BCD, or ASCII data file from which (write) or into which (read) the processor transfers data. Enter this file address without the # symbol.

Important: You cannot use indirect addresses for the data file address in a BTR or BTW instruction.

- **Length** is the number of data file words to read/write.

If You Set the
Length to:

The Processor:

0

Reserves 64 words for block transfer data. The block transfer module transfers the maximum words it can handle.

1 to 64

Transfers the number of words specified.

Important: A floating point element consists of two words; when you specify a value in the Length field for a floating point data file, only half of those floating point elements are read/written. For example, if you specify 64 for the length, 32 floating point elements will actually be read/written.

Important: Floating point data file lengths must be an even number.

- **Continuous** determines the mode of operation.

If You Specify:	The Instruction Uses This Mode:
Yes	Continuous – once the rung goes true, the instruction continues to transfer data until the continuous (.CO) bit is reset and the rung is false or you edit the instruction and specify NO for continuous mode.
No	Non-continuous – the instruction is enabled each time the rung goes true and performs only one data transfer per rung transition.

Using Status Bits

To use the BTR and BTW instructions correctly, examine the instruction's status bits stored in the control block. These bits are in word 0 of the control block.



ATTENTION: Except for the continuous bit .CO (bit 11) and the timeout bit .TO (bit 08), do not modify any status bit while the block transfer instruction is enabled. Unpredictable operation could occur with possible damage to equipment and/or injury to personnel.

Important: The bit labels (.EN, .ST, .CO, etc.) can only be used with the block transfer file type (BT).

This Bit:	Is Set:
Enable .EN (bit 15)	when the rung goes true. This bit shows that the instruction is enabled (that the block transfer is in progress). In non-continuous mode, the .EN bit remains set until the block transfer finishes or fails and the rung goes false. In continuous mode, once the .EN bit is set, it remains set regardless of the rung condition.
Start .ST (bit 14)	when the processor begins transferring data. The .ST bit is reset at the false-to-true transition after the .DN bit or .ER bit is set.
Done .DN (bit 13)	at completion of the block transfer, if the data is valid. The .DN bit is set asynchronous to the program scan so the .DN bit may go true any time after the block transfer is initiated. The .DN bit is reset the next time the associated rung goes from false to true.

This Bit:	Is Set:
Error .ER (bit 12)	when the processor detects that the block transfer failed. The .ER bit is reset the next time the associated rung goes from false to true.
Continue .CO (bit 11)	when you edit the instruction for repeated operation of the block transfer after the first scan, independent of whether the processor continues to scan the rung. Reset the .CO bit if you want the rung condition to initiate block transfers (return to non-continuous mode). If you are using continuous block transfers in a sequential function chart, see Appendix B, "SFC Reference," in this manual.
Enable-waiting .EW (bit 10)	when the block transfer request enters the queue. If the queue is full, this bit remains reset until there is room in the queue. The .EW bit is reset when the associated rung goes from false to true. In continuous mode, once the .EW bit is set, it remains set. Use the .EW bit to verify that a BTW or BTR instruction is queued before leaving an SFC step.
No Response .NR (bit 09)	if the block transfer module does not respond to the first local block transfer request. The .NR bit is reset when the associated rung goes from false to true (not used with remote block transfer).
Time Out .TO (bit 08)	if you reset the time out bit through ladder logic or data monitor, the processor repeatedly tries to send a block transfer request to an unresponsive module for four seconds before setting the .ER bit. If you set the .TO bit through ladder logic or data monitor, the processor disables the four-second timer and requests a block transfer one more time before setting the .ER bit.
Read-Write .RW (bit 07)	controlled by the instruction. A 0 indicates a write operation; a 1 indicates a read operation.



ATTENTION: The processor runs block transfer instructions asynchronously to program scan. The status of these bits could change at any point in the program scan. If you examine these bits in ladder logic, copy the status once to a storage bit whose status is synchronized with the program scan. Otherwise, timing problems may invalidate your program with possible damage to equipment or injury to personnel.

Important: When using integer (N) and block transfer (BT) file types, the .EN, .ST, .DN, .ER, .EW, and .NR bits are cleared during prescan.

Your ladder program should condition the use of block transfer data on the state of the .DN bit.

Using the Control Block

In addition to the status bits, the control block contains other parameters the processor uses to control block transfer instructions. Table 15.F lists these values.

Table 15.F
Values in the Block Transfer Control Block

Word – Integer Control Block	BT Control Block	Description
0	.EN through .RW	Status bits
1	.RLEN	Requested word count
2	.DLEN	Transmitted word count / error code (Enhanced PLC-5 processors)
3	.FILE	File type / number
4	.ELEM	Element number

Requested Word Count (.RLEN)

This is the number of words to be transferred between the processor and the module (0-64 words); the processor creates a file of the length you specified that starts at the data address you enter. The length depends on the target module or your application. For example, if you specify 30 in this field, you are specifying a block length of 30 and the processor creates a 30-word file; if you specify 64, you are specifying a block length of 64 and the processor creates a 64-word file. If you specify a 0 when you enter the block transfer instruction, the processor lets the block transfer module determine the number of words that need to be transferred and creates a default 64-word file.

Transmitted Word Count (.DLEN)

This is the number of words the module actually transferred after the instruction completes execution. The processor uses this number to verify the transfer. This number should match the requested word count (unless the transmitted word count is zero). If these numbers do not match, the processor sets the .ER bit (bit 12).

The Enhanced PLC-5 processors also have error codes (word 2 of the integer file control block or stored in the .DLEN word of the BT control block) that the processor can set during the transfer. If a block transfer error occurs in an Enhanced PLC-5 processor, the error code is stored in the transmitted word count. This error can be identified by its negative number. Only one error code is stored at a time (a new error code overwrites any previous error code). Table 15.G lists these error codes.

Table 15.G
Enhanced PLC-5 Processor Block Transfer Error Codes

Error Number:	Description:
-1	not used
-2	not used
-3	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.
-4	There was an invalid transfer of block transfer write data between the adapter and the block transfer module.
-5	The checksum of the block transfer read data was wrong.
-6	The block transfer module requested a different length than the associated block transfer instruction. This could happen if a 64-word block transfer instruction was executed and the block transfer module's default length was not 64 words.
-7	Block transfer data was lost due to a bad communication channel. Possible reasons are noise, bad connections, and loose wires. Check resistors.
-8	Error in block transfer protocol – unsolicited block transfer.
-9	The block transfer timeout, set in the instruction, timed out before completion.
-10	No communication channels are configured for remote I/O or rack number does not appear in rack list.
-11	No communication channels are configured for the requested rack or slot.
-12	The adapter is faulted or not present for the BT command.
-13	Queues for remote block transfers are full.

File Number (.FILE)

This number identifies the file number of the integer file from which the data is written or to which the data is read. For example, the file number of N7:20 is 7.

Element Number (.ELEM)

This number identifies the starting word in the data file address. For example, in N7:20, the word number is 20.

Selecting Continuous Operation

Continuous block transfer is similar to discrete I/O transfer in that the I/O is updated continuously, but continuous block transfer updates block transfer I/O, such as analog input and analog output data.

Continuous mode lets you perform multiple block transfers by programming only one block transfer instruction (with no input conditions on the rung). Once the continuous block transfer starts, the transfer is continuously executed once per scan, independent of whether the processor continues to scan the associated rung and independent of the rung condition. To enable continuous operation, select Continuous when you enter the block transfer instruction.

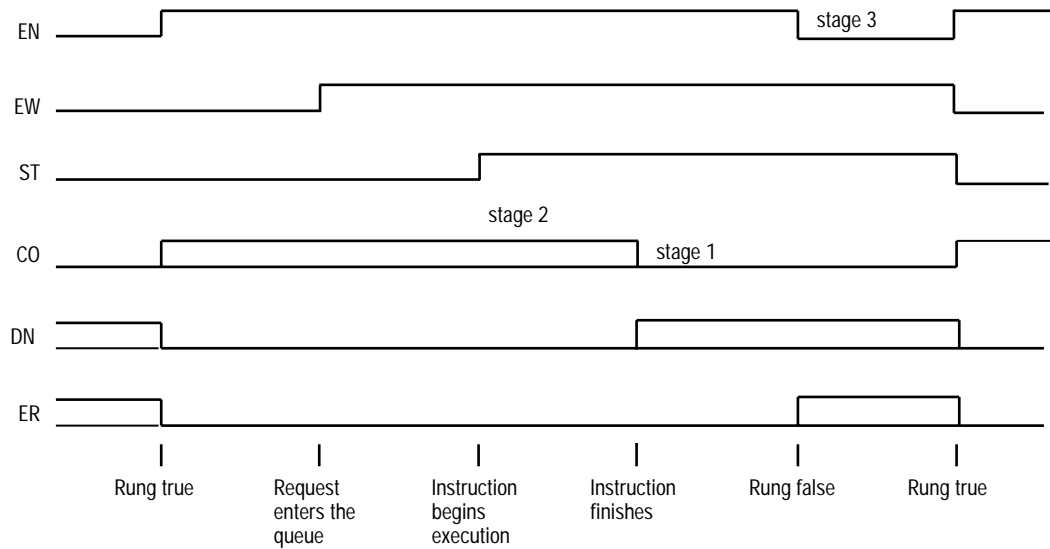
Continuous mode works as follows (Figure 15.3):

1. When the rung that contains the block transfer instruction goes true, the processor sets the .EN bit. The processor also resets the .DN, .ER, .ST, .EW and .NR bits.
2. The processor then queues the block transfer request. When the block transfer request enters the queue, the processor sets the .EW bit.
3. When the processor starts to process the block transfer request, the processor sets the .ST bit.
4. If no error occurs during transmission, the processor sets the .DN bit. The processor copies the actual number of elements sent or received by the block transfer instruction into the transmitted word count (word 2 of the control block).

If an error occurs, the processor sets the .ER bit. If an error occurs in an Enhanced PLC-5 processor, the processor also puts the error code in the transmitted word count location as a negative number.

5. If there is no response (and after the processor sets the .NR bit), the processor tries to send the block transfer again. If the .TO bit is reset, the processor repeatedly sends the request for four seconds. If the .TO bit is set, the processor only retries the request one time.
6. If a continuous block transfer has an error, you must restart it to continue. (See Figure 15.7 on page 15-18 for an example program.)

Figure 15.3
Timing Diagram for Status Bits in Continuous BTR and BTW Instructions



Stage 1 - If .CO set, return to stage 2; if .CO reset, go to stage 3

Stage 2 - Return here for continuous operation

Stage 3 - Go here if .CO is reset

A continuous block transfer continues as long as the processor stays in Run or Test mode and the transfer does not error. If you switch to Program mode or the processor faults, the block transfer stops and will not start again until the processor scans the rung that contains the block transfer instruction. If running continuous block transfers from within sequential function charts, see appendix B.

To stop continuous operation either: modify the block transfer instruction and select non-continuous, or reset the .CO bit.

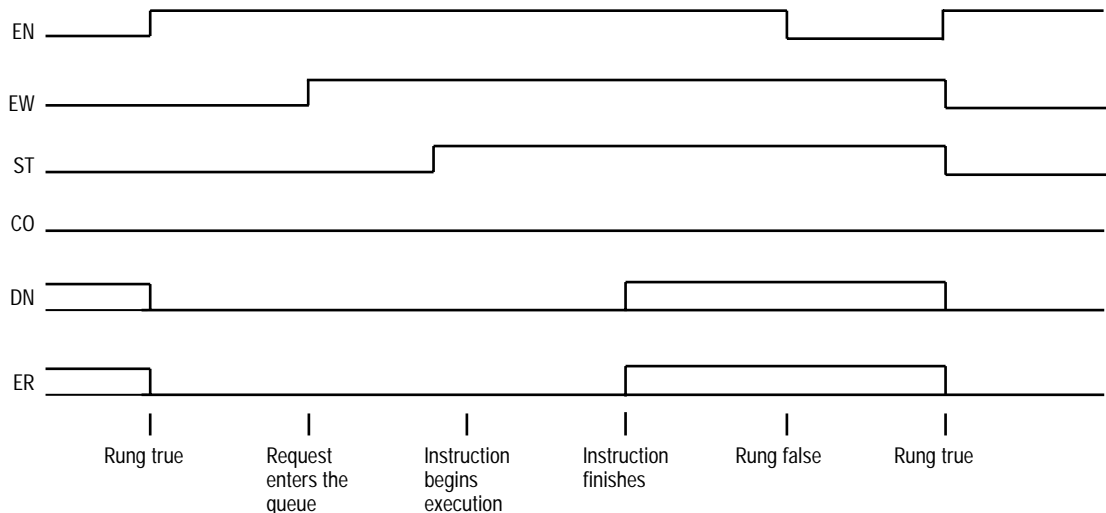
Selecting Non-Continuous Operation

Non-continuous block transfer updates block transfer I/O one time when the rung goes true. A non-continuous block transfer maintains block integrity. The entire block of data is updated each time the processor runs the block transfer instruction. Use non-continuous mode when you want to control when the block transfer occurs or the number of times the block transfer occurs.

Non-continuous mode works as follows (Figure 15.4):

1. When the rung that contains the block transfer instruction goes true, the processor sets the .EN bit. The processor also resets the .DN, .ER, .ST, .EW and .NR bits.
2. The processor then queues the block transfer request. When the block transfer request enters the queue, the processor sets the .EW bit.
3. When the processor starts to process the block transfer request, the processor sets the .ST bit.
4. If no error occurs during transmission, the processor sets the .DN bit after the block transfer instruction finishes. If an error occurs, the processor sets the .ER bit.
5. This signifies that one block transfer finished. The next time the rung goes false, the processor resets the .EN bit.

Figure 15.4
Timing Diagram for Status Bits in Non-Continuous BTR and BTW Instructions



Block Transfer Timing – Classic PLC-5 Processors

The time to complete a block transfer in a Classic PLC-5 processor depends on:

- instruction run time
- waiting time in the queue
- transfer time

Instruction Run Time

The time in microseconds it takes the processor to execute a block transfer instruction is defined by these formulas:

Write:

$$310 + 11.2Q + 5.4W$$

Read:

$$250 + 11.2Q$$

Where:

Represents:

Q	number of queued block transfer requests to the same I/O chassis with the continuous bit set
---	----------------------------------------------------------------------------------------------

W	number of words to transfer
---	-----------------------------

Waiting Time in the Queue

The waiting time in the queue is the sum of the transfer times yet to occur before the block transfer request (for which you are calculating time) to the same I/O chassis.

Transfer Time

The transfer time in milliseconds between the active buffer and the module starts when the processor sets the start bit and ends when the processor sets the done bit. The transfer time is defined by these formulas:

Write:

local $0.9 + 0.1W$

remote (57.6K baud) $13 + 30C + 0.3W$

Read:

local $0.9 + 0.1W$

remote (57.6K baud) $9 + 21.3C + 0.3W$

Where:

Represents:

C	number of full remote logical racks
---	-------------------------------------

W	number of words to transfer
---	-----------------------------

Block Transfer Timing – Enhanced PLC-5 Processors

The time to complete a block transfer in Enhanced PLC-5 processors depends on:

- instruction run time
- waiting time in the holding area (queue)
- transfer time

Instruction Run Time

The time it takes the processor to execute a block transfer instruction is the same for a read or a write: 450 microseconds.

Waiting Time in the Holding Area

The waiting time in the holding area is the sum of the transfer times yet to occur before the block transfer request (for which you are calculating time) to the same I/O chassis.

Transfer Time

The transfer time in milliseconds between the active buffer and the module starts when the processor sets the start bit and ends when the processor sets the done bit. The transfer time is defined by this formula (same for a read or write):

$$\text{local} \quad 600 \mu\text{sec} + x(w)$$

$$\text{remote (57.6K baud)} \quad 4 + 8C + 0.3W$$

$$\text{remote (115K baud)} \quad 4 + 4.6C + 0.15W$$

$$\text{remote (230K baud)} \quad 4 + 3.2C + 0.075W$$

Where this:	Represents:
x	<ul style="list-style-type: none"> • 8 or less block transfers queued in local rack = 86 μsec • more than 8 block transfers queued in local rack = 300 μsec <p>Note: This timing assumes that no other block transfers are queued to the same slot and that successive block transfers to the same slot are executed every 1000 μsec.</p>
C	number of full remote logical racks
W	number of words to transfer

Programming Examples

Program your processor for block transfer using one of the following methods based on your application requirements (Table 15.H):

Table 15.H
Block Transfer Programming Methods

If You Want to:	Use this Method:
Program block transfers to and from the same module when you want the order of execution to follow the same order scanned in the program	Bidirectional alternating
Continuously repeat bidirectional alternating block transfers and the step will be scanned	Bidirectional alternating repeating
Program block transfers to and from the same module when you want the transfers to continue regardless of which SFC steps are active	Bidirectional continuous*
Program a BTR from, or a BTW to a module when you want the block transfer to execute based on an event	Directional non-continuous
Continuously repeat a block transfer and the step will be scanned	Directional repeating
Program a BTR from or BTW to a module when you want the transfer to continue regardless of which SFC steps are active	Directional continuous*
Ensure block integrity	Buffering block transfer data

* Only use continuous mode when you want a block transfer to continue executing even when the logic which controls it is not being scanned.

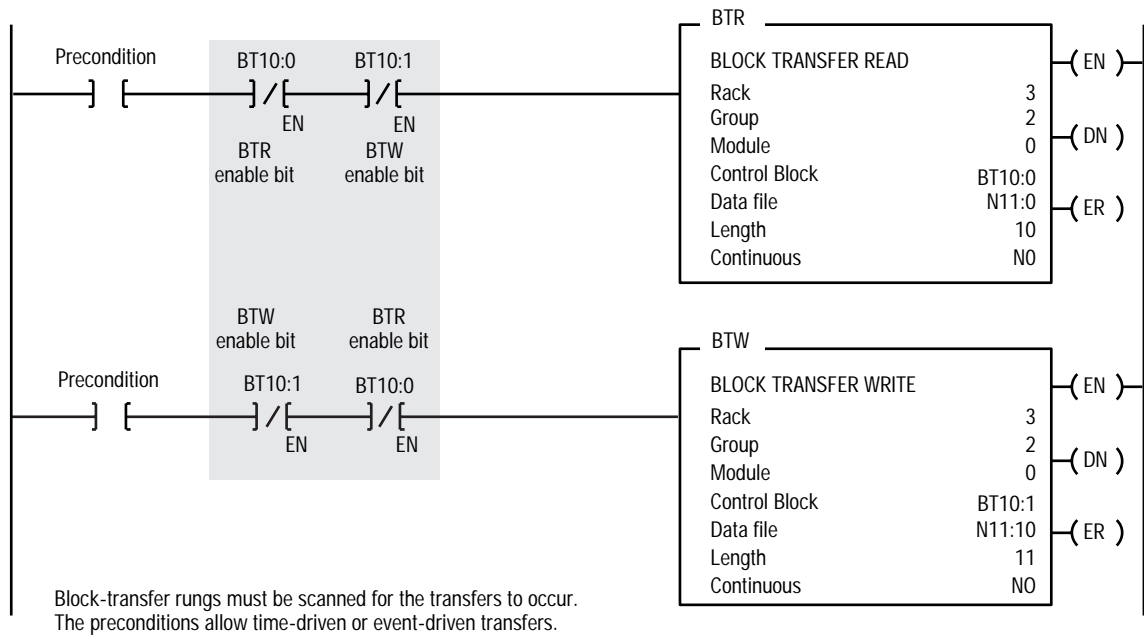
Important: These examples show an Enhanced PLC-5 processor using the BT file type. If you are using a Classic PLC-5 processor, substitute an appropriate integer file.

Example Bidirectional Alternating Block-Transfer

Figure 15.5 shows a bidirectional alternating block transfer example. Using rungs like this example makes sure the block transfer requests are executed in the order in which they were sent to the queue. The processor alternates between the BTRs and BTWs in the order in which they are scanned by virtue of the XIO condition. The XIO condition prevents the block transfer read and block transfer write from queuing simultaneously. The block transfer continues as long as the optional condition is true.

On the rungs of logic, you may include as many optional conditions as you wish to the left of the required enable bit condition (XIO) transition.

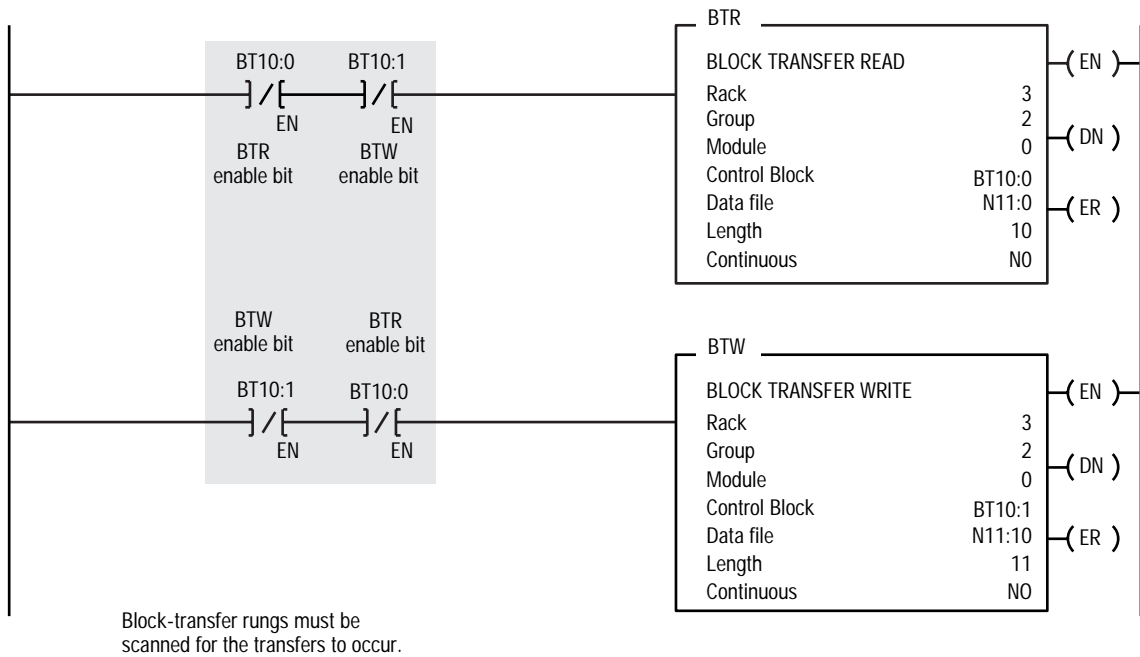
Figure 15.5
Example Bidirectional Alternating Block Transfer



Example Bidirectional Alternating Repeating Block-Transfer

Figure 15.6 shows a bidirectional alternating repeating block transfer example. Using rungs like this example makes sure the block transfer requests are executed in the order in which they were sent to the queue. The processor alternates between the BTRs and BTWs in the order in which they are scanned by virtue of the XIO conditions. The XIO conditions prevents the block transfer read and block transfer write from queueing simultaneously. The block transfers continue as long as the step is scanned.

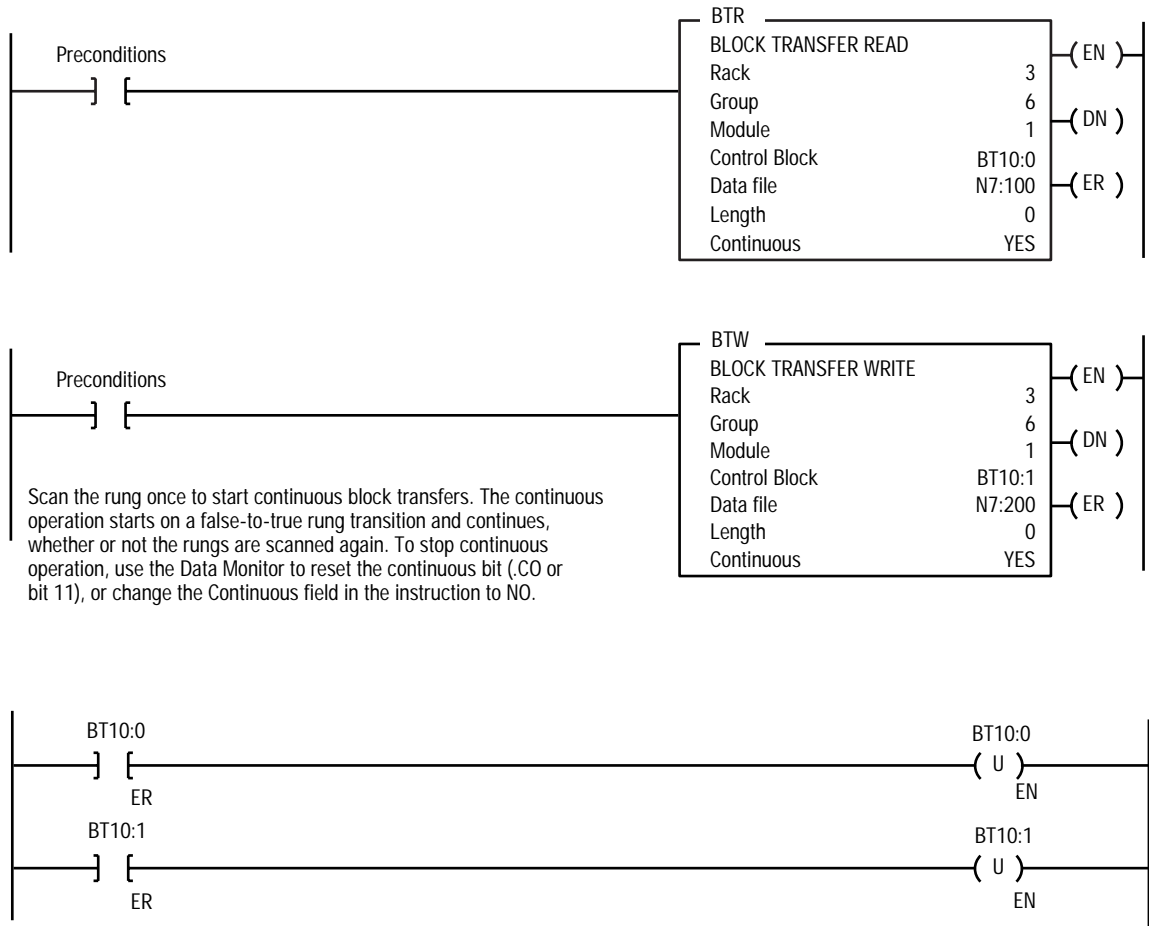
Figure 15.6
Example Bidirectional Alternating Repeating Block Transfer



Example Bidirectional Continuous Block-Transfer

Figure 15.7 shows a bidirectional continuous block transfer example.

Figure 15.7
Example Bidirectional Continuous Block Transfer



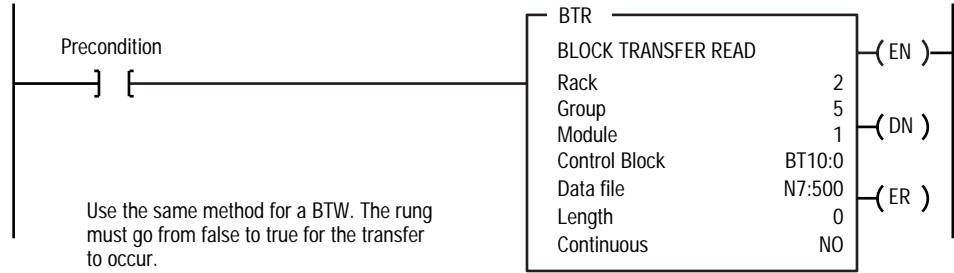
Scan the rung once to start continuous block transfers. The continuous operation starts on a false-to-true rung transition and continues, whether or not the rungs are scanned again. To stop continuous operation, use the Data Monitor to reset the continuous bit (.CO or bit 11), or change the Continuous field in the instruction to NO.

These rungs will reset block transfers and should be placed in logic where rungs are being scanned for error recovery. Your logic must rescan the block transfers with preconditions true in order to restart continuous block transfers.

Example Directional Non-Continuous Block-Transfer

Figure 15.8 shows a directional non-continuous block transfer example. The block transfer executes once for every false-to-true transition of the precondition.

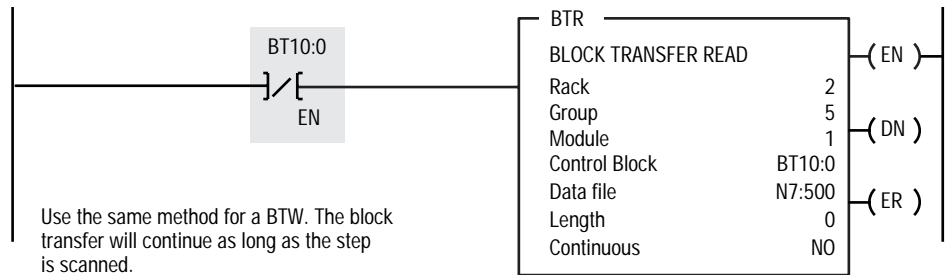
Figure 15.8
Example Directional Non-Continuous Block Transfer



Example Directional Repeating Block Transfer

Figure 15.9 shows a directional repeating block transfer example.

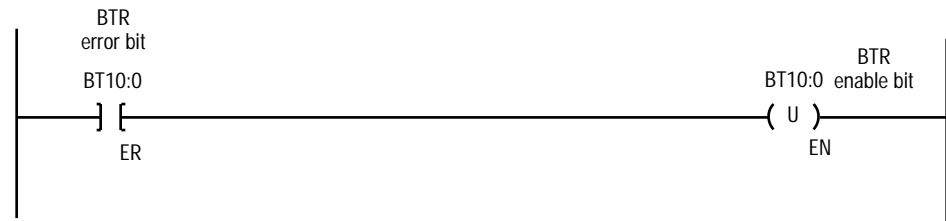
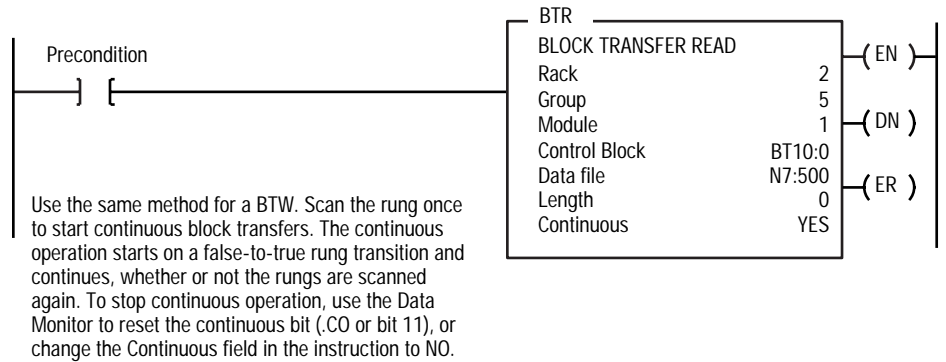
Figure 15.9
Example Directional Repeating Block Transfer



Example Directional Continuous Block-Transfer

Figure 15.10 shows a directional continuous block transfer example.

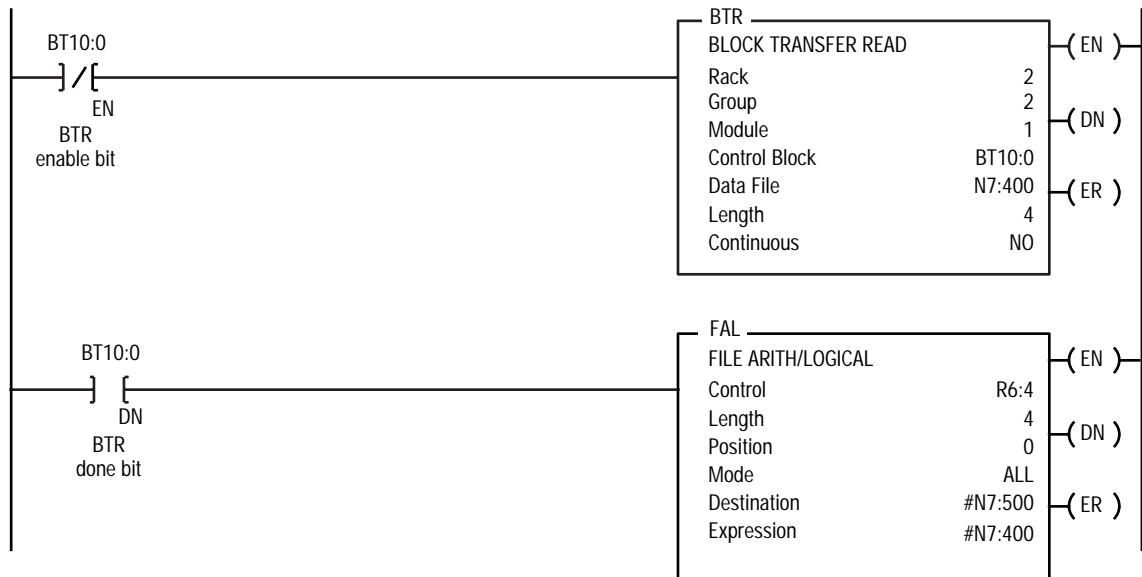
Figure 15.10
Example Directional Continuous Block Transfer



This rung will reset block transfers and should be placed in logic where rungs are being scanned for error recovery. Your logic must rescan the block transfers with preconditions true in order to restart continuous block transfers.

Example Buffering Block Transfer-Data

If you block transfer data repeatedly, buffer the file by examining the BTR done bit and executing a file-to-file move (or copy) when the done bit is true. This ensures file integrity of the block transfer read data file.



ControlNet I/O Transfer (CIO) Instruction



Using the CIO instruction, you can perform ladder-initiated unscheduled transfers (up to 64 elements) to I/O modules (typically analog or intelligent) on a ControlNet network. For more information on ControlNet I/O operations, see the ControlNet PLC-5 Programmable Controllers User Manual.

When the input conditions go from false to true, data is transferred according to the instruction parameters you set when entering the CIO instruction.

To program a CIO instruction, you must provide the ControlNet PLC-5 processor with a control block address, which contains the status and instruction parameters. After you enter the control block parameters, the programming terminal displays an instruction entry screen from which you enter instruction parameters stored in the control block address.

Control Block Address

With ControlNet PLC-5 processors, use a ControlNet transfer (CT) file type for the control block. For example, CT12:1 is a valid CIO control block address.

Important: You cannot use indirect addresses for the control block address in a CIO instruction.

After you enter the control block address for the CIO instruction, the programming terminal displays an instruction entry screen.

Using the CIO Instruction

You can use the CIO instruction to transfer up to 64 elements of data (per CIO instruction) over a ControlNet link. The instruction entry screen for the CIO instruction lets you configure the following information (Table 15.I).

Important: The PLC-5 structured text programming software does not support the CIO instruction.

Table 15.I
CIO Instruction Entry Screen Configuration

If You Want to:	Press this Key:
Change the command type. Toggle among the following: <ul style="list-style-type: none"> • 1771 Read selects a block transfer read. • 1771 Write selects a block transfer write. • 1794 Fault Action selects the action the module takes when the adapter faults and the connection is terminated. • 1794 Idle Action selects the action the module takes when the connection is idle. • 1794 Config Data changes the configuration for the 1794 module. • 1794 Safe State Data changes the value of the safe state data for the 1794 module. 	[F1] – Command Type
Enter a PLC-5 data table address of the ControlNet processor.	[F2] – PLC-5 Address
Enter the size in elements. Type the number of elements and press [Enter]. <ul style="list-style-type: none"> • 1 (1794 Fault Action and 1794 Idle Action) • 1-15 (1794 Config Data and 1794 Safe State Data) • 0-64 (1771 Read and 1771 Write) Note: If you enter 0 for 1771 Read and/or 1771 Write, 64 words are reserved for block transfer.	[F3] – Size in Elements
Enter the destination network address. Type a number (1-99) and press [Enter].	[F8] – Local Node
Enter the destination slot number. Type a number and press [Enter]. <ul style="list-style-type: none"> • 0-7 (1794 command types) • 0-15 (1771 command types) Note: The slot number represents the physical slot in the chassis occupied by the module. To find your slot number, count from the left I/O slot starting with 0.	[F9] – Slot Number

After entering and accepting the rung containing the CIO instruction, the data monitor screen for the CIO instruction lets you display the parameters for the control block of the current CIO instruction. From the data monitor screen, you can define the following parameters (Table 15.J).

Table 15.J
CIO Instruction Control Block Parameters

If You Want to:	Press this Key:
Toggle the control bit that the cursor is on. You can toggle among the TO, EW, CO, ER, DN, ST, and EN bits.	[F2] – Toggle Bit
Change the size of the block of data to send or receive.	[F3] – Size in Elements
Change the address for which the data is displayed.	[F5] – Specify Address
Display the data table values for the next file.	[F7] – Next File
Display the data table values for the previous file.	[F8] – Previous File
Display the data table values for the next element.	[F9] – Next Element
Display the data table values for the previous element.	[F10] – Previous Element

Using Status Bits

The CIO instruction uses the following status bits:

This Bit:	Is Set:
Enable .EN (bit 15)	when the rung goes true. The .EN bit is reset when the .DN bit or .ER bit is set. This bit shows that the instruction is enabled.
Start .ST (bit 14)	when the processor begins executing the CIO instruction. The .ST bit is reset when the .DN bit or .ER bit is set.
Done .DN (bit 13)	when the last word of the CIO instruction transferred. The .DN bit is reset the next time the associated rung goes from false to true. The .DN bit is only active in non-continuous mode.
Error .ER (bit 12)	when the processor detects that the message transfer failed. The .ER bit is reset the next time the associated rung goes from false to true.
Continue .CO (bit 11)	manually for repeated operation of the CIO instruction after the first scan, independent of whether the processor continues to scan the rung.
Enable-Waiting .EW (bit 10)	when the processor detects that a message request entered the queue. The processor resets the .EW bit when the .ST bit is set.
Time Out .TO (bit 08)	through ladder logic to stop processing the message. The processor sets the .ER bit.



ATTENTION: The processor controls status bits .ST and .EW asynchronously to the program scan. If you examine these bits in ladder logic, copy the status to a storage bit whose status is synchronized with the program scan. Otherwise, timing problems may invalidate your program with possible damage to equipment or injury to personnel.



ATTENTION: For continuous mode to operate correctly, you must set the .CO bit (either on the configuration screen or through ladder logic) **before** you enable the CIO instruction.

Using the CT Control Block

In addition to the status bits, the CT control block contains these parameters that the ControlNet PLC-5 processor uses to control CIO instructions.

Word:	CT Control Block:	Description:
0	.EN through .TO	Status bits See "Using Status Bits" above.
1	.ERR	Error code This is where the processor stores the error code if a problem occurs during message transmission.
2	.RLEN	Requested length This is the requested number of elements you wish to transfer with the message instruction.
3	.DLEN	Done length This is the number of elements the module actually transferred after the instruction completes execution. This number should match the requested length (unless the requested length is 0).
4	.FILE	File number This number identifies the file number of the file from which the data is written or to which the data is read. For example, the file number of N12:1 is 12.
5	.ELEM	Element number This number identifies the starting word in the data file address. For example, in N12:1, the word number is 1.

Notes:

Message Instruction MSG

Using the Message Instruction

The message instruction (MSG) is used to read or write a block of data to another station on the DH+ link, to an attached Control Coprocessor, to the VMEbus using a VME PLC-5 processor, or to another node on an Ethernet network. The MSG instruction is also used to create unscheduled messages initiated by one ControlNet PLC-5 processor and sent to another ControlNet PLC-5 processor and to allow Enhanced PLC-5 (other than Ethernet PLC-5) processors to program and upload/download unsolicited messages over Ethernet through the PLC-5 Ethernet Interface Module. You program the MSG instruction in ladder logic.

The MSG instruction over DH+ can communicate with PLC-2[®], PLC-3[®], PLC-5[®], PLC-5/250[™], and SLC 5/03[™] and SLC-5/04[™] processors on local or remote links.

For more information on the operands (and valid data types/values of each operand) used by the MSG instruction, see Appendix C.

Message (MSG)

Description:



The MSG instruction transfers up to 1000 elements of data (120 words using a Control Coprocessor); the size of each element depends on the data table section you specify and the type of message command you use. For example, one binary element contains one 16-bit word and one floating point element contains two 16-bit words.

The MSG instruction transfers data in packets. Each DH+ data packet can contain up to 120 words. If your message transfer contains more words than fit in one packet, the transfer requires more than one packet of transfer data. The more packets of data to transfer, the longer the transfer takes. Over Ethernet each packet can be up to 709 words, thus making it a more efficient networking option.

The following table lists which Enhanced PLC-5 processors (series and revision) you can use with the MSG instruction to transfer data from/to a PLC-5 processor or to/from an SLC 5/03 or 5/04 processor in SLC native mode.

Processor Series/Revision:	Processors:
Series A / revision M	PLC-5/40, -5/40L, -5/60, -5/60L
Series A / revision J	PLC-5/30
Series A / revision H	PLC-5/11, -5/20
Series B / revision J	PLC-5/40, -5/40L, -5/60, -5/60L
Series C / revision G	Enhanced, Ethernet, and VME PLC-5 processors
Series C / revision H	ControlNet PLC-5
Series D / revision A	Enhanced, Ethernet, ControlNet, and VME PLC-5 processors

Entering Parameters

Specify a control block address when you first enter the MSG instruction. The control block is where all of the information relating to the message will be stored. After entering the control block, the programming terminal automatically displays a data entry screen, from which you enter instruction parameters that are stored at the control block address. You can also use the data monitor screen for the MSG instruction to edit selected parameters.

Control Block Address

With Classic PLC-5 processors, use an integer file (N) without the # symbol for the message control block. For example N7:0 is a valid MSG control block address.

If You Have this Processor:	Use this Control Block Address:
Classic PLC-5	an integer file (N) without the # symbol for the message control block. Example: N7:0
Enhanced PLC-5, Ethernet PLC-5, or VME PLC-5	<p>an integer file (N) or the message (MG) file type to access the message control block for DH+ transfers. Example: MG10:0</p> <p>Using the MG control block, the control block size is fixed at 56 words. This size is displayed on the screen in the BLOCK SIZE field. You must use the MG control block if you are sending messages to an SLC 500 processor using the SLC read and write commands, or if you are sending message out any port other than channel 1A.</p>
Ethernet PLC-5, ControlNet PLC-5, VME PLC-5	a message (MG) file type to access the VMEbus, Ethernet, or ControlNet network

You cannot use indirect addresses for the control block address in an MSG instruction. If you have an MSG instruction created with version 3.21 or earlier that uses a control block with an indirect address, you must delete the instruction and re-enter it without an indirect address.

For the VME PLC-5 processors to do transfers to the VMEbus, the MSG instruction must be programmed with an MG control block.

For ControlNet PLC-5 processors to perform transfers on the ControlNet network, the MSG instruction must be programmed with an MG data type in the control block.

The control block size varies according to the length of the message; if you communicate with a PLC-2 processor, the control file will be approximately 11 or 12 words. If you communicate with a PLC-3, PLC-5 or PLC-5/250 processor, the control file will be approximately 11 to 15 words. This size is displayed on the screen in the BLOCK SIZE field.

For Enhanced PLC-5 processors, you can use an integer file (excluding ControlNet PLC-5 processors) or message (MG) file type to access the message control block for DH+ transfers. For example, MG10:0 is a valid MSG control block address for Enhanced PLC-5 processors. Using the MG file type, the control block size is fixed at 56 words; this size is displayed on the screen in the BLOCK SIZE field.

For the Ethernet PLC-5 processors, a MSG instruction going through port 2, the Ethernet port, uses two consecutive message elements (i.e., MG10:0 and MG10:1). Your programming software might display a warning when you select port 2.

MSG Data Entry Screen

After you enter the control block address for a MSG instruction, the programming software automatically displays a data entry screen for the MSG instruction using the appropriate data type (integer or message). Press the function key for the data you want to modify. You can specify the following MSG parameters from the data entry screen:

This Function Key:	Specifies this Information:
[F1] – Command Type	Whether the MSG instruction performs a read or write operation and to what type of processor the message is going to.
[F2] – PLC-5 Address	The data file address of the processor containing the message instruction. If the MSG operation is write, this address is the starting word of the source file. If the MSG operation is read, this address is the starting word of the destination file.
[F3] – Size in Elements	The number of elements (1-1000) to be transferred.

This Function Key:	Specifies this Information:
[F4] – Local/Remote	<p>LOCAL: the message is sent to a device on the local DH+ link. REMOTE: the message is sent through a bridge (DH, DH II, etc.) to another DH+ link.</p> <p>If you select REMOTE, the function keys [F5] – Remote Station, [F6] – Link ID, and [F7] – Remote Link are active.</p>
[F5] – Remote Station	<p>The DH or DH II address (1-376 octal) of the target station.</p> <p>PLC-2 and PLC-3 processors require communication adapter modules (1771-KA2 and 1775-KA, respectively) when they operate as stations on data highway. In these configurations, the remote station address is the address of the communication adapter module.</p>
[F6] – Link ID	The remote link where the processor you want to communicate with resides. The default is 0.
[F7] – Remote Link	Toggles through DH, DH II, and other choices for what connects the remote link to the local DH+.
[F8] – Local Node	<p>The local station address on the DH+ (0-77 octal).</p> <p>If you communicate with another processor on the local link, this address is the address of the target station on the local link.</p> <p>If you communicate with a target station on a remote link, this address is the station number of the communication adapter module that bridges the data highway.</p>
[F9] – Destination Address	The starting address of the source or destination file in the target processor.
[F10] – Port Number	The channel for message communications. Valid options are: 0, 1A (default), 1B, 2A, 2B, and 3A for the ASCII command.

If you select the ASCII option using the **[F1] – Command Type** key, (for use with PLC-5/V40 doing read/writes to the VMEbus), the software displays a new screen for you to enter the text for your ASCII communications. Refer to the PLC-5/VME VMEbus Programmable Controllers User Manual for the syntax of command text to do VMEbus transfers.

For Control Coprocessor data transfers using the MSG instruction, use the following choices:

- communication command – select PLC-3 word range read or PLC-3 word range write
- destination data table address – 00 through 31, matches corresponding read/write handler in coprocessors application program
- Port number – 3A

Using the Message Instruction for Ethernet Communications

The message (MSG) instruction transfers up to 1000 elements of data; the size of each element depends on the data table section that you specify and the type of message command that you use. One binary element contains one 16-bit word, for example, and one floating-point element contains two 16-bit words.

The MSG instruction transfers data in packets. Each packet can contain up to 709 words for Ethernet processors. If your message transfer contains more words than fit in one packet, the transfer requires more than one packet of transfer data. The more packets of data to transfer, the longer the transfer takes.

Entering Parameters

The control block is where all of the information relating to the message is stored. Ethernet message instructions use two consecutive MSG elements—the first one contains the message information, the second one contains the destination address.

Important: Since the Ethernet messages need two consecutive control blocks, the message control block that you specify must start on an even number.



ATTENTION: While configuring MSG instructions for the DH+ and serial links, keep in mind the files used for Ethernet MSG control blocks.

If you choose a file being used as an Ethernet control block, the programming software prompts you to choose whether you want to overwrite the file. If you choose to overwrite the file, unpredictable machine operation could occur.

After entering the control block, the programming terminal automatically displays a data entry screen, from which you enter instruction parameters that are stored at the control block address.

You must enter a port number of 2 to enable a special screen for Ethernet transfers.

This Field:	Specifies this Information:
Command Type	Whether the MSG instruction performs a read or write operation. The software toggles between: <ul style="list-style-type: none"> • PLC-5 Typed Read • PLC-5 Typed Write • PLC-5 Typed Write to SLC • PLC-5 Typed Read from SLC • SLC Typed Logical Read • SLC Typed Logical Write • PLC-2 Unprotected Read • PLC-2 Unprotected Write • PLC-3 Word Range Read • PLC-3 Word Range Write • ASCII
PLC-5 Address	The data file address of the processor containing the message instruction. If the MSG operation is write, this address is the starting word of the source file. If the MSG operation is read, this address is the starting word of the destination file.
Size in Elements	The number of elements (1-1000) to be transferred.
IP Address	The MSG instruction's destination node. <ul style="list-style-type: none"> • If the destination is another PLC-5/20E, -5/40E, or -5/80E, the destination must be a full Internet address • If the destination is an INTERCHANGE™ client program, enter the word "CLIENT" as the destination node. Do not enter an IP address. <p>Note: You must set [F10] – Port Number to 2 in order to access this function.</p>
Destination Address	The starting address of the source or destination file in the target processor.
Port Number	The channel for message communications. Ethernet communications use channel 2.
Multihop	Select yes if you want to send the MSG instruction to a ControlLogix device. Then use the Multihop tab to specify the path of the MSG instruction. For more information, see "Configuring an Ethernet Multihop MSG Instruction" on page 16-9.

The Ethernet PLC-5 processors do not support **hostnames** as a means of addressing messages. However, you can use host names with PLC-5 programming software for connecting to Ethernet PLC-5 processors if a name server is on the network or a host file is maintained on your workstation.

Using the Message Instruction for PLC-5 Ethernet Interface Module Communications

Use the MSG instruction to allow Enhanced PLC-5 processors to program and upload/download unsolicited messages (up to 1000 elements each) over Ethernet through the PLC-5 Ethernet Interface Module. The size of each element depends on the data table section that you specify and the type of message command that you use. One binary element contains one 16-bit word, for example, and one floating-point element contains two 16-bit words.

To program a MSG instruction, you must provide the PLC-5 Ethernet Interface Module and the connected Enhanced PLC-5 processor with a control block address, which contains the status and instruction parameters. After entering the control block parameters, the programming terminal displays an instruction entry screen from where you can enter instruction parameters that are stored in the control block address.

Entering Parameters

The control block is where all of the information relating to the message is stored. Ethernet message instructions use two consecutive MSG elements—the first one contains the message information, the second one contains the destination address.

Important: Since the Ethernet messages need two consecutive control blocks, the message control block that you specify must start on an even number.

After entering the control block, the PLC-5 programming software automatically displays a data entry screen, from which you enter instruction parameters that are stored at the control block address.

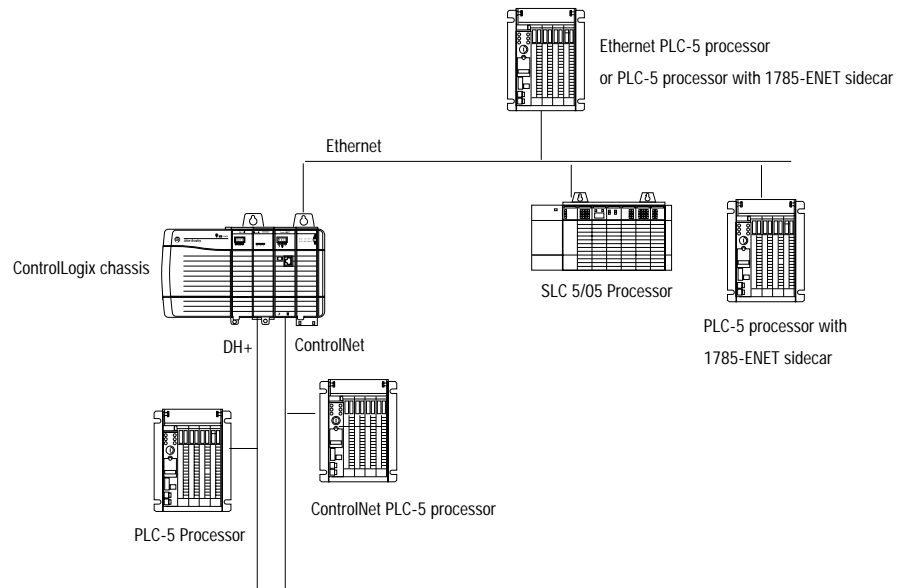
You must enter a port number of 3A to enable a special screen for transfers over Ethernet with the PLC-5 Ethernet Interface Module.

This Field:	Specifies this Information:
Command Type	Whether the MSG instruction performs a read or write operation. The software toggles between: <ul style="list-style-type: none"> • PLC-5 Typed Read • PLC-5 Typed Write • PLC-5 Typed Write to SLC • PLC-5 Typed Read from SLC • SLC Typed Logical Read • SLC Typed Logical Write • PLC-2 Unprotected Read • PLC-2 Unprotected Write • PLC-3 Word Range Read • PLC-3 Word Range Write • ASCII
PLC-5 Address	The data file address of the processor containing the message instruction. If the MSG operation is write, this address is the starting word of the source file. If the MSG operation is read, this address is the starting word of the destination file.
Size in Elements	The number of elements (1-1000) to be transferred.
Host/IP Address	The MSG instruction's destination node. <ul style="list-style-type: none"> • If the destination is an Enhanced PLC-5 processor, the destination must be a full Internet address • If the destination is an INTERCHANGE client program, enter the word "CLIENT" as the destination node. Do not enter an IP address. <p>Note: You must set [F10] – Port Number to 2 in order to access this function.</p>
Destination Address	The starting address of the source or destination file in the target processor.
Port Number	The channel for message communications. The PLC-5 Ethernet Interface Module communications use channel 3A.

Removal of the PLC-5 Ethernet Interface Module will not change the format of the MSG instructions defined for the module.

Configuring an Ethernet Multihop MSG Instruction

The series E, revision D and later PLC-5 processors can communicate over Ethernet with ControlLogix devices or through a ControlLogix Ethernet (1756-ENET) module to other PLC-5 processors. You need either an Ethernet PLC-5 processor or any PLC-5 processor with a series A, revision E 1785-ENET sidecar module. The following diagram shows an Ethernet PLC-5 processor and the other PLC and SLC processors it can communicate with using a multihop MSG instruction.



To communicate through a ControlLogix 1756-ENET module, you configure the multihop feature of a MSG instruction from the Ethernet PLC-5 processor (or PLC-5 processor with 1785-ENET sidecar module) to the target device. You need RSLogix 5 programming software. Enable the multihop option when you specify the target device. Then use the Multihop tab to specify the path of the MSG instruction.

If you want to go through the ControlLogix 1756-ENET module and out the 1756-DHRIO module to the target device, you:

- use Gateway configuration software to configure the 1756-DHRIO module routing table in the ControlLogix system.
- specify a Link ID number on channel properties for channel 2/3A of the Ethernet PLC-5 processor (or PLC-5 processor with a 1785-ENET sidecar module).

For more information about configuring a PLC-5 channel and specifying the path of the MSG instruction, see the documentation for your programming software.

Using the Message Instruction for ControlNet Communications

Use the MSG instruction to create unscheduled messages (up to 1000 elements each) that are initiated by one ControlNet PLC-5 processor and sent to another ControlNet PLC-5 processor. For more information on ControlNet I/O operations, see the ControlNet PLC-5 Programmable Controllers User Manual.

When the input conditions go from false to true, data is transferred according to the instruction parameters you set when entering the MSG instruction.

To program a MSG instruction, you must provide the ControlNet PLC-5 processor with a control block address, which contains the status and instruction parameters. After entering the control block parameters, the programming terminal displays an instruction entry screen from where you can enter instruction parameters that are stored in the control block address.

Control Block Address

With ControlNet PLC-5 processors, use a message data file (MG) for the message control block. For example, MG20:50 is a valid MSG control block address.

You can use the Message (MG) file type and the MSG instruction to send two commands over ControlNet within the local ControlNet link:

- PLC-5 Typed Write
- PLC-5 Typed Read

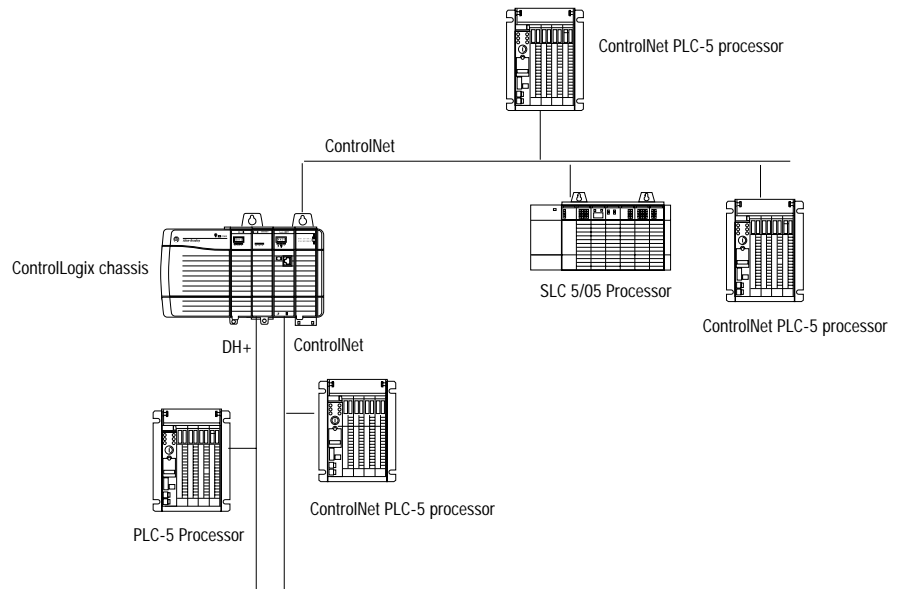
After you enter the control block address for the MSG instruction, the programming terminal displays an instruction entry screen. Press the function key for the data you want to modify. You can specify the following from the instruction entry screen:

This Field:	Specifies this Information:
Command Type	Change the command type. Toggle between the following: <ul style="list-style-type: none"> • PLC-5 Typed Write selects a write operation to a ControlNet PLC-5 processor • PLC-5 Typed Read selects a read operation from another ControlNet PLC-5 processor
PLC-5 Address	The PLC-5 data table address of the ControlNet processor. If the MSG operation is write, this address is the starting word of the source file. If the MSG operation is read, this address is the starting word of the destination file.
Size in Elements	The number of elements (1-1000) to be transferred.
Local Node	The destination node address (1-99).
Destination Address	The starting address of the source or destination file in the target processor.
Port Number	The channel for message communications. The port number must be 2 for ControlNet.
Multihop	Select yes if you want to send the MSG instruction to a ControlLogix device. Then use the Multihop tab to specify the path of the MSG instruction. For more information, see "Configuring an ControlNet Multihop MSG Instruction" on page 16-11.

Configuring a ControlNet Multihop MSG Instruction

The series F, revision A and later ControlNet PLC-5 processors can communicate over ControlNet with ControlLogix devices or through a ControlLogix ControlNet (1756-CNB) module to other PLC-5 processors on other networks. Earlier series ControlNet PLC-5 processors can be updated to support ControlNet-to-ControlNet network messages and to respond to multihop messages over a DH+ network. The series F, revision A ControlNet PLC-5 processors add support for ControlNet-to-other network messages.

The following diagram shows an ControlNet PLC-5 processor and the other PLC and SLC processors it can communicate with using a multihop MSG instruction.



To communicate through a ControlLogix 1756-CNB module, you configure the multihop feature of a MSG instruction from the ControlNet PLC-5 processor to the target device. You need RSLogix 5 programming software. Enable the multihop option when you specify the target device. Then use the Multihop tab to specify the path of the MSG instruction.

If you want to go through the ControlLogix 1756-ENET module and out the 1756-DHRIO module to the target device, you:

- use Gateway configuration software to configure the 1756-DHRIO module routing table in the ControlLogix system.
- specify a Link ID number on channel properties for channel 2/3A of the Ethernet PLC-5 processor (or PLC-5 processor with a 1785-ENET sidecar module).

For more information about configuring a PLC-5 channel or specifying the path of the MSG instruction, see the documentation for your programming software.

Using Status Bits

The MSG instruction uses the following status bits:



ATTENTION: Do not modify any status bit while the Instruction is enabled.

Unpredictable machine operation could occur with possible damage to equipment and/or injury to personnel.

Important: The bit labels (.EN, .ST, .CO, etc.) can only be used with the message file type (MG).

This Bit:	Is Set:
Enable .EN (bit 15)	when the rung goes true. This bit shows that the instruction is enabled (that the instruction is being executed). In non-continuous mode, .EN bit remains set until the message is completed and the rung goes false. In continuous mode, once .EN bit is set, it remains set regardless of the rung condition.
Start .ST (bit 14)	when the processor begins executing the MSG instruction. The .ST bit is reset when the .DN bit or .ER bit is set.
Done .DN (bit 13)	when the last packet of the MSG instruction transferred. The .DN bit is reset the next time the associated rung goes from false to true. The .DN bit is only active in non-continuous mode.
Error .ER (bit 12)	when the processor detects that the message transfer failed. The .ER bit is reset the next time the associated rung goes from false to true.
Continue .CO (bit 11)	manually for repeated operation of the MSG instruction after the first scan, independent of whether the processor continues to scan the rung. Reset the .CO bit if you want the rung condition to initiate messages (return to non-continuous mode).
Enable-Waiting .EW (bit 10)	when the processor detects that a message request entered the queue. The processor resets the .EW bit when the .ST bit is set.
No Response .NR (bit 09)	if the target processor does not respond to the first MSG request. The .NR bit is reset when the associated rung goes from false to true.
Time Out .TO (bit 08)	if you set the .TO bit through ladder logic, the processor stops processing the message and sets the .ER bit (timeout error 55). A DH+ message time out in 30-60 seconds. A ControlNet message will time out in 4 seconds
No Cache .NC (ControlNet processors only)	if you set the .NC bit, the open connection is closed when the MSG is done. If this bit remains reset, the connection remains open even when the MSG is complete.



ATTENTION: The processor controls status bits .ST and .EW asynchronously to the program scan. If you examine these bits in ladder logic, copy the status to a storage bit whose status is synchronized with the program scan. Otherwise, timing problems may invalidate your program with possible damage to equipment or injury to personnel.

Important: If the SFC startover and .CO bits are cleared, the .EN, .ST, .DN, .ER, .EW, and .NR bits are cleared during prescan.

Using the Control Block

In addition to the status bits, the control block contains other parameters the processor uses to control message instructions. Table 16.A lists these values.

Table 16.A
Values in the Control Block

Word – Integer Control Block	Message Control Block	Description
0	.EN thru .RW	Control bits
0 - low byte	.ERR	Error code
2 - high byte	.RLEN	Requested length
2 - low byte	.DLEN	Done length
3		Internal data

Error Code (.ERR)

This is where the processor stores the error code if a problem occurs during message transmission. Error codes are listed in Table 16.E.

Requested Length (.RLEN)

This is the requested amount of elements the user wishes to transfer with the message instruction.

Transmitted Length (.DLEN)

This is the number of elements the module actually transferred after the instruction completes execution. This number should match the requested length.

Entering Parameters

Communication Command

The following table describes the communication commands.

If You Want the Instruction to:	Select the Command:
Read data identified by a type code. This command reads data structures without you specifying the actual word length. For example, if you choose a typed read of the PLC-5 timer data section with a requested data size of 5 elements, the MSG instruction reads 15 words (5 timer structures of 3 words each).	PLC-5 Typed Read
Write data identified by a type code. This command writes data structures without you specifying the actual word length.	PLC-5 Typed Write
Read 16-bit words from any area of the PLC-2 data table or PLC-2 compatibility file.	PLC-2 Unprotected Read
Write 16-bit words to any area of the PLC-2 data table or PLC-2 compatibility file.	PLC-2 Unprotected Write
Read data identified by a type code. This command reads data structures without specifying the actual word length. This command provides additional data verification for communications between a PLC-5 and SLC 500 processor. ¹	PLC-5 Typed Read from SLC ^{2, 3}
Write data identified by a type code. This command writes data structures without specifying the actual word length. This command provides additional data verification for communications between a PLC-5 and SLC 500 processor. ¹	PLC-5 Typed Write to SLC ^{2, 3}
Read a range of words, starting at the address specified for the external address in the MSG control file and reading sequentially the number of words specified for the requested size field in the MSG control file. The data that is read is stored, starting at the address specified for the internal address in the MSG control file. This is used for communicating between a PLC-5 and SLC 500 processor. ¹	SLC Typed Logical Read ³
Write a range of words, starting at the address specified for the internal address in the MSG control file and writing sequentially the number of words specified for the requested size field in the MSG control file. The data from the internal address is written, starting at the address specified for the external address in the MSG control file. This is used for communicating between a PLC-5 and SLC 500 processor. ¹	SLC Typed Logical Write ³
Read a range of words, starting at the address specified for the external address in the MSG control file and reading sequentially the number of words specified for the requested size field in the MSG control file. The data that is read is stored, starting at the address specified for the internal address in the MSG control file.	PLC-3 Word Range Read
Write a range of words, starting at the address specified for the internal address in the MSG control file and writing sequentially the number of words specified for the requested size field in the MSG control file. The data from the internal address is written, starting at the address specified for the external address in the MSG control file.	PLC-3 Word Range Write

¹The PLC-5 is limited to a maximum message of 103 words (206 bytes). The maximum message size for SLC 5/03™ and SLC 5/04™ processors is 103 words (206 bytes). The maximum message size capability of all other SLC 500 processors is 41 words (82 bytes).

²These commands are valid only with any SLC 5/04 and SLC 5/03 series C and later processors.

³These commands are only valid with processors listed in the table on page 16-2.

You can use the Typed Read and Typed Write commands to transfer data table sections without counting the actual words per data table element. You only have to specify the number of elements you want to transfer. For example, in the data table timer section, one element contains 3 words, but in the binary data table section, one element contains one word.

External Data Table Addresses

The following table lists the valid external data table addresses.

This Communication Command:	To this Device:	Requires that You Enter:	Example Address:
PLC-5 Typed Read	PLC-5/250	the address in double quotes	"1N0:0"
PLC-5 Typed Write	PLC-5	the address	N7:0
	1775-S5	the address in double quotes, precede with a \$ character	"\$N7:0"
	1775-SR5		
PLC-2 Unprotected Read PLC-2 Unprotected Write	PLC-2 PLC-2 compatibles	octal number of 16-bit word offset	025
PLC-3 Word Range Read PLC-3 Word Range Write	PLC-5/250	the address in double quotes	"1N7:0"
	PLC-5	the address in double quotes, precede with a \$ character	"\$N7:0"
	1775-S5	the address in double quotes, precede with a \$ character, or just the address (which will be slightly faster)	"\$N7:0" N7:0
	1775-SR5		
	1771-DMC Control Coproductors	the address in double quotes "00" to "31" to match C program	"01"
SLC Typed Logical Read SLC Typed Logical Write	SLC 500 processors	the address	N7:0
PLC-5 Typed Read to SLC PLC-5 Typed Write from SLC	SLC 5/03 and 5/04 processors	the address	N7:0

PLC-2 to PLC-5 Compatibility Files

In order to send a message between a PLC-2 and a PLC-5, you must use a PLC-2 compatibility file within the PLC-5 processor. This file number must be the decimal equivalent of the octal address of the PLC-2. We recommend that the octal address of the PLC-2 be greater than 10 so that it does not interfere with the default PLC-5 data files.

For example, if a PLC-2 is at station 12, any message it sends to a PLC-5 defaults to file 10 in the PLC-5 (decimal equivalent to octal 12). Also note that since the PLC-2 addresses is octal, if you have a PLC-2 address as 024 in a write command, the write actually occurs in the PLC-5's word 20 (decimal equivalent to octal 24).

Sending SLC Typed Logical Read and Typed Logical Write Commands

Follow these guidelines when programming SLC Typed Logical Read and SLC Typed Logical Write commands:

- You must use the MG data type for the MSG control block.
- PLC-5 Data Table Address and the Destination address types should match when the data type is supported by the PLC-5 and SLC 5/03 and 5/04 processors. If you want to send a data type that the SLC 5/03 or 5/04 processors do not support, the SLC processors interpret that data as integer. This table maps the data types from the PLC-5 processors to the SLC 5/03 and 5/04 processors.

This PLC-5 Data Type:	Is Interpreted by the SLC 5/03 and 5/04 Processors as:
Binary (B)	Bit
Integer (N)	Integer
Output (O)	Integer
Input (I)	Integer
Status (S)	Integer
ASCII (A)	ASCII
BCD (D)	Integer
SFC status (SC)	Integer
String (ST)	String
BT control (BT)	Integer
ControlNet transfer (CT)	Integer
Timer (T)	Timer
Counter (C)	Counter
Control (R)	Control
Float (F)	Float
MSG control (MG)	Integer
PID control (PD)	Integer

- To read/write from the SLC input, output (read only), or status file, specify an integer PLC-5 Data Table Address and specify the address of the SLC input, output, or status file. For example, S:37 for word 37 of the SLC status file. Specify SLC input/output addresses by logical format, i.e., O:001 references slot 1.

- PLC-5 ASCII data is byte data (1/2 word); whereas, an SLC ASCII data element is one word. Therefore, if you request an PLC 5 Typed Read of 10 elements, the SLC 500 processor sends a packet containing 20 bytes (10 words).
- PLC-5 processors allows 1000 elements for most data types whereas SLC 500 processors only allow 256 elements.

Monitoring a Message Instruction

To monitor or edit MSG instruction parameters and status bits after you enter the MSG instruction, display the data monitor screen for the MSG instruction and file type you are using.

If You Are Using this File Type:	See:
integer (N)	Table 16.B
message (MG)	Table 16.C

If you are using an integer (N) file type, you can do the following from the data monitor screen (Table 16.B):

Table 16.B
Data Monitor Screen for the MSG Instruction – N File Type

If You Want to:	Press this Key:
specify the number of elements (1-1000) you want to read from or write to the network station	[F3] – Size in Elements
set and reset status bits	[F9] – Toggle Bit

If you are using an message (MG) file type, you can do the following from the data monitor screen (Table 16.C):

Table 16.C
Data Monitor Screen for the MSG Instruction – MG File Type

If You Want to:	Press this Key:
Toggle the control bit that the cursor is on. You can toggle among the TO, NR, EW, CO, ER, DN, ST, and EN bits	[F2] – Toggle Bit
Change the size of the block of data to send or receive.	[F3] – Size in Elements
Change the address for which the data is displayed.	[F5] – Specify Address
Display the data table values for the next file.	[F7] – Next File
Display the data table values for the previous file.	[F8] – Previous File
Display the data table values for the next element.	[F9] – Next Element
Display the data table values for the previous element.	[F10] – Previous Element

Selecting Continuous Operation

Continuous mode lets you use multiple message transfers by programming only one MSG instruction (with no input conditions on the rung). Once the continuous message transfer starts, the transfer is continuously executed, independent of whether the processor continues to scan the associated rung and independent of the rung condition. To enable continuous operation, set the .CO bit.



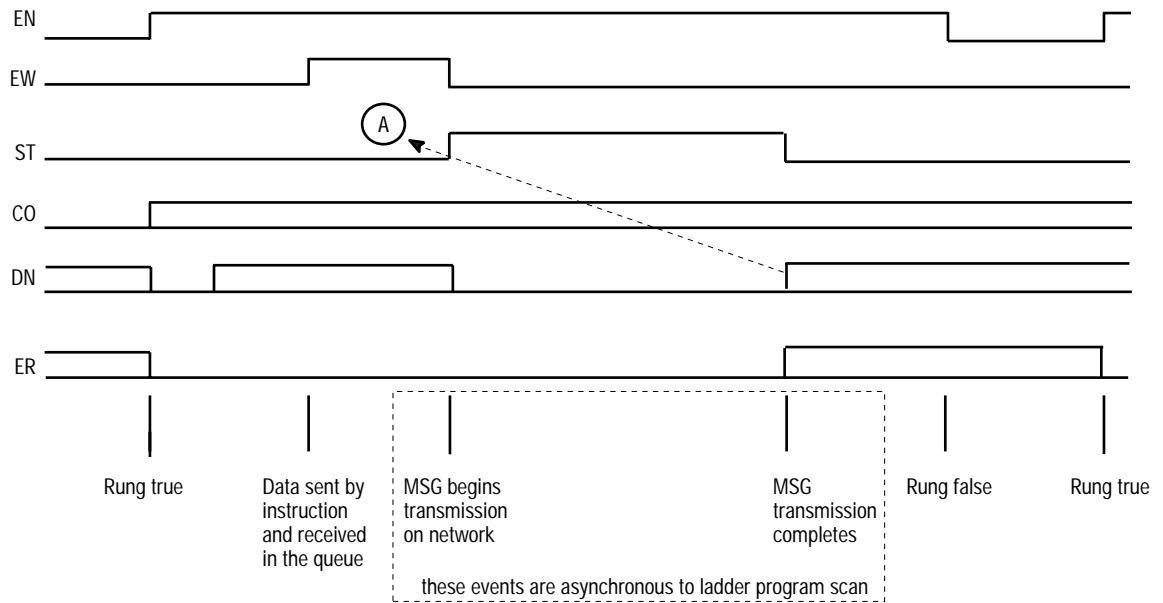
ATTENTION: For continuous mode to operate correctly, you must set the .CO bit (either on the configuration screen or through ladder logic) **before** you enable the MSG instruction.

Continuous mode works as follows (Figure 16.1):

1. When the rung that contains the MSG instruction goes true, the processor initiating the MSG instruction sets the .EN bit. The processor also resets the .ER and .DN bits.
2. The processor then queues the message request. When the message request enters the queue, the processor sets the .EW bit.
3. When the processor starts to process the message request, the processor sets the .ST bit. The next time the processor receives network control, the processor transmits the message.
4. If an error occurs, the processor sets the .ER bit and stores an error code in the lower byte of word 0 of the control block for Classic PLC-5 processors and word 1 of the control block for Enhanced PLC-5 processors.

Important: Figure 16.1 is appropriate for Enhanced PLC-5 processors only. You can reset Classic PLC-5 processors by toggling the error or enable bits.

Figure 16.1
Timing Diagram for Status Bits in Continuous MSG Instructions



A When the MSG transmission completes, the cycle starts over here without rung transitions

A continuous message transfer continues as long as the processor stays in Run or Test mode. If you switch to Program mode or the processor faults, the message transfer stops and will not start again until the processor scans the rung that contains the MSG instruction.

To stop continuous operation, reset the .CO bit.

Earlier PLC-5 processors, prior to series E, reset the .EN bit of a continuous MSG whenever the rung is scanned false and the .DN or .ER bit is set. Series E and later processors leave the .EN bit set when the rung is false and the .DN bit is set. This indicates the true state of the MSG instruction, which is still operating. However, if the rung is false and the .ER bit is set, the .EN bit is reset. This lets you restart an errored continuous MSG instruction by toggling the state of the rung.

Selecting Non-Continuous Operation

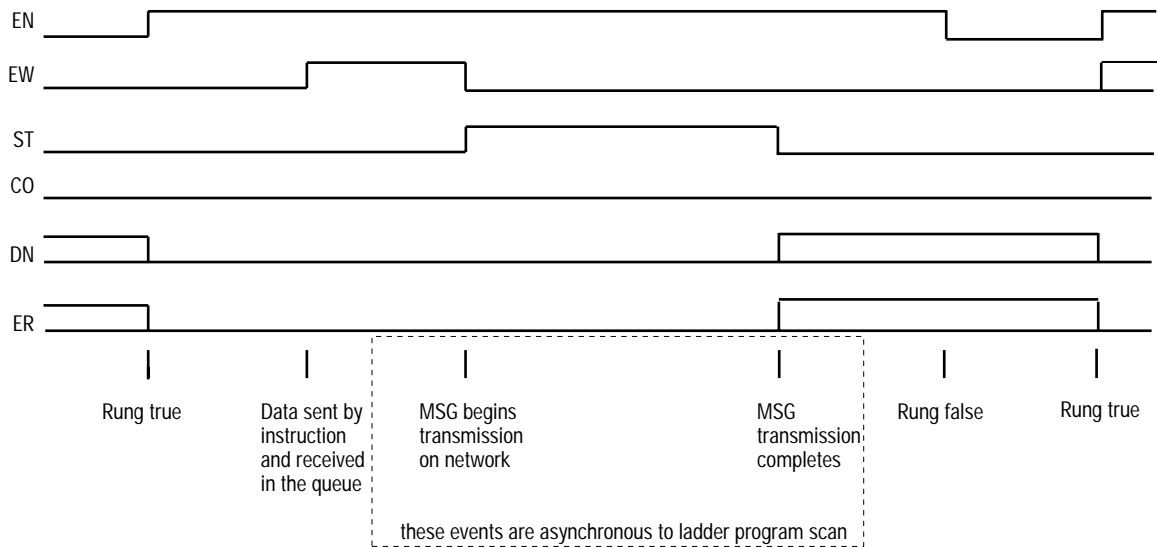
Non-continuous mode performs the message transfer one time for each false-to-true transition of the rung that contains the MSG instruction. Non-continuous operation occurs as long as the .CO bit remains reset. Use non-continuous mode when you want to control when the message transfer occurs or the number of times the message transfer occurs.

Non-continuous mode works as follows (Figure 16.2):

1. When the rung that contains the MSG instruction goes true, the processor initiating the MSG instruction sets the .EN bit. The processor also resets the .DN and .ER bits.
2. The processor then queues the message request. When the message request enters the queue, the processor sets the .EW bit.

3. When the processor starts to process the message request, the processor sets the .ST bit. The next time the processor receives network control, the processor transmits the message.
4. If no errors occur during transmission, the processor sets the .DN bit and resets the .ST bit after the last packet in the first execution of the MSG instruction transfers. If an error occurs, the processor sets the .ER bit, resets the .ST bit, and stores an error code in the lower byte of word 0 of the control block for Classic PLC-5 and word 1 of the control block for Enhanced PLC-5 processors.
5. The next time the rung goes false, the processor resets the .EN bit. Then when the associated rung goes true again, the message transfer cycle starts again.

Figure 16.2
Timing Diagram for Status Bits in Non-Continuous MSG Instructions



MSG Timing

The time required for one PLC-5 processor to send or receive a message to/from another processor on the DH+ link depends on the number of:

- stations on the DH+ link
- messages transmitted from active stations
- bytes of data of all transmitted messages
- message requests already in the queue

Timing starts with setting the enable bit and ends with setting the done bit in the ladder program of the station initiating the message. The order of operation is shown in Table 16.D.

Table 16.D
Message Instruction Operation

Receiving MSG (Station A reading/receiving from Station B)	Sending MSG (Station A writing/sending to Station B)
station A enables the message instruction in the ladder program	station A enables the message instruction in the ladder program
station A obtains the token and transmits the read command (station B acknowledges immediately)	station A obtains the token and transmits data (station B acknowledges immediately)
station B obtains the token and transmits requested data	station B stores the data in memory
station A receives the data and acknowledges immediately	station B obtains the token to respond that the write is complete
station A sets the done bit	station A sets the done bit when it receives a response

You can estimate the time (in milliseconds) for transmitting one packet over DH+ using the following formulas:

Processor Type: **Formula:**

Classic PLC-5	Message time = TP + TT + OH + P + 8 (number of messages)
---------------	----------------------------------------------------------

Enhanced PLC-5	Message time = TP + TT + OH + 8 (number of messages)
----------------	------------------------------------------------------

where:

TP = token pass = (1.5) (1 + number of stations on DH+ link)

TT = transmission time = (0.28) (number of data words)
Number of data words in all transmitted messages for one token pass around the DH+ link.

OH = DH+ overhead = 20ms

P = longest program scan for any processor on the DH+ link (application value in milliseconds)

See the PLC-5VMEbus Programmable Controllers User Manual and the Ethernet PLC-5 Programmable Controllers manual for performance numbers and benchmarks.

Error Codes

When the processor detects an error during the transfer of message data, the processor sets the .ER bit and enters an error code that you can monitor from your programming terminal. If the message is non-continuous, the processor sets the .ER bit the first time the processor scans the MSG instruction.

Table 16.E
Errors Detected By the Processor

Code:			Description (Displayed on the Data Monitor screen):
Enhanced PLC-5 ¹ MG data type	Classic PLC-5 ² N data type	Ethernet Only	
0037	55	0037	message timed out in local processor
0083	131	0083	processor is disconnected
0089	137	0089	message buffer is full If the MSG is going out channel 0, there are not enough internal buffers available. Decrease the number of MSG instructions to this port. Otherwise, the destination node sent back a MSG indicating that its buffers are full. Decrease the number of MSG instructions going to the destination node.
0092	146	0092	no response (regardless of station type)
00D3	211	00D3	you formatted the control block incorrectly
00D5	213	00D5	incorrect address for the local data table
0200	2		link layer timed out or received a NAK
0300	3		duplicate token holder detected by link layer
0400	4		local port is disconnected
0500	5		application layer timed out waiting for a response
0600	6		duplicate node detected
0700	7		station is off line
0800	8		hardware fault
1000	129	1000	illegal command from local processor
2000	130	2000	communication module not working
3000	131		remote node is missing, disconnected, or shut down
4000	132	4000	processor connected but faulted (hardware)
5000	133	5000	you used the wrong station number
6000	134	6000	requested function is not available
7000	135	7000	processor is in program node

¹Hexadecimal – word 1 of the control block

²Decimal – low byte of word 0 of the control block

Code:			
Enhanced PLC-5 ¹ MG data type	Classic PLC-5 ² N data type	Ethernet Only	Description (Displayed on the Data Monitor screen):
8000	136	8000	processor's compatibility file does not exist
9000	137	9000	remote node cannot buffer command
B000	139	B000	processor is downloading so it is inaccessible
F001	231	F001	processor incorrectly converted the address
F002	232	F002	incomplete address
F003	233	F003	incorrect address
F006	236	F006	addressed file does not exist in target processor
F007	237	F007	destination file is too small for number of words requested
F00A	240	F00A	target processor cannot put requested information in packets
F00B	241	F00B	privilege error, access denied
F00C	242	F00C	requested function is not available
F00D	243	F00D	request is redundant
F011	247	F011	data type requested does not match data available
F012	248	F012	incorrect command parameters
0010 ³		0010	no IP address configured for the network
0011 ³		0011	already at maximum number of connections
0012 ³		0012	invalid Internet address or host name
0013 ³		0013	no such host
0014 ³		0014	cannot communicate with the name server
0015 ³		0015	connection not completed before user-specified timeout
0016 ³		0016	connection timed out by the network
0017 ³		0017	connection refused by destination host
0018 ³		0018	connection was broken
0019 ³		0019	reply not received before user-specified timeout
001A ³		001A	no network buffer space available
		F01A	file owner active – the file is being used
		F01B	program owner active – someone is downloading, online editing, or set the program owner with APS in the WHO Active screen

¹Hexadecimal – word 1 of the control block

²Decimal – low byte of word 0 of the control block

³Errors detected by a Enhanced PLC-5 processor attached to a PLC-5 Ethernet Interface Module only.

Table 16.F
Errors Detected By the VME Processor

PLC-5/40V (hexadecimal – word 1 of the control block)	Description (Displayed on the Data Monitor screen):
0000	success
0001	invalid ASCII message format
0002	invalid file type
0003	invalid file number
0004	invalid file element
0005	invalid VME address
0006	invalid VME transfer width
0007	invalid number of elements requested for transfer
0008	invalid VME interrupt level
0009	invalid VME interrupt status-id value
000A	VMEbus transfer error (bus error)
000B	unable to assert requested interrupt (already pending)
000C	raw data transfer setup error
000D	raw data transfer crash (PLC switched out of run mode)
000E	unknown message type (message type not ASCII)

ASCII Instructions ABL, ACB, ACI, ACN, AEX, AIC, AHL, ARD, ARL, ASC, ASR, AWA, AWT

Using ASCII Instructions Enhanced PLC-5 Processors Only

The ASCII instructions read, write, compare and convert ASCII strings. These instructions are only supported by Enhanced PLC-5 processors. Table 17.A lists the available ASCII instructions.

Table 17.A
Available ASCII Instructions

If You Want to:	Use this Instruction:	On Page:
See how many characters are in the buffer, up to and including the end of line character	ABL	17-4
See how many total characters are in the buffer	ACB	17-5
Convert a string to an integer value	ACI	17-6
Concatenate two strings into one	ACN	17-7
Extract a portion of a string to create a new string	AEX	17-7
Configure your modem handshake lines	AHL	17-8
Convert an integer value to a string	AIC	17-9
Read characters from the buffer and put into a string	ARD	17-10
Read one line of characters from the buffer and put into a string	ARL	17-12
Search a string for another string	ASC	17-14
Compare two strings	ASR	17-15
Write a string with user-configured characters appended	AWA	17-15
Write a string	AWT	17-17

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

There are two types of ASCII instructions:

Type of ASCII Instruction:	Description:
ASCII port control	read, write, set/reset handshake lines, examine the length of the buffer (ARD, ARL, AWT, AWA, AHL, ACB, ABL)
ASCII string	manipulate string data, such as compare, search, extract, concatenate, convert to/from integer (ASR, ASC, AEX, ACN, ACI, AIC)

ASCII instructions are dependent upon one another. For example, if you have an ARD (ASCII read instruction) and then an AWT (ASCII write), the done bit on the ARD must be set before the AWT can begin executing (even if the AWT was enabled while the processor was executing the ARD). A second ASCII instruction cannot begin until the first has completed. However, the processor does not wait for an ASCII instruction to complete before continuing to execute your ladder program (non-ASCII instructions).

Using Status Bits

You can examine status bits in the ladder program to examine some event. The processor changes the states of status bits as the processor executes the instruction. You address the status bits by mnemonic (or bit number) in the control element address.

The ASCII instructions use the length (.LEN) and position (.POS) fields in some instructions as well as the following status bits:

Description:	Explanation of Status Bit:
Found.FD (08)	Reserved
Unload.UL (10)	This bit may be used by the user to cancel an ASCII read or write in progress. The time out may occur immediately or up to 6 seconds later.
Error.ER (11)	The instruction did not complete successfully. Note: If this bit is set, the .EN bit is cleared and the .DN bit is set during prescan.
Synchronous Done.EM (12)	The bit is set on the first scan of the instruction after it is completed.
Asynchronous Done.DN (13)	The bit is set immediately upon successful completion of the instruction, asynchronous to program scan. Note: If this bit is set, the .EN bit is cleared and the .DN bit is set during prescan.
Queue.EU (14)	The bit is set when the instruction is successfully queued.
Enable.EN (15)	The bit is set when the rung goes true and is reset after completion of the instruction and the rung goes false. Note: If this bit is set and the .DN and .ER bits are cleared, the control word is cleared during prescan.

Using the Control Block

In addition to the status bits, the control block contains other parameters the processor uses to control ASCII transfer instructions. Table 17.B lists these values.

Table 17.B
Values in the Control Word

Word – Integer Control Block	ASCII Control Block	Description
0	.EN, .DN, etc	Status Bits
1	.LEN	Word Length
2	.POS	Character Position

Length (.LEN)

This is the number of characters the operation is to be performed on.

Position (.POS)

This is the current character number which the operation has executed.

Using Strings

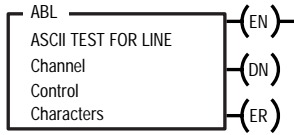
You can address string lengths by adding a .LEN to any string address (for example, ST17:1.LEN).

String lengths must be between 0 and 82 bytes. In general, lengths that are outside of this range cause the processor to set a minor fault (S:17/8) and the instruction is not executed.

Important: You configure append or end-of-line characters on the Channel Configuration screen. The default append characters are carriage return and line feed; the default end-of-line (termination) character is carriage return. For more information, see your software user manual.

Test Buffer for Line (ABL)

Description:



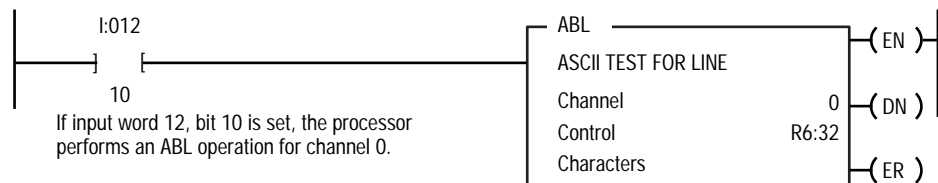
Use the ABL instruction to see how many characters are in the buffer, up to and including the end-of-line characters (termination). On a false-to-true transition, the system reports the number of characters in the Position field, and sets the done bit. The serial port must be in User mode.

Entering Parameters

To use the ABL instruction, you must supply this information:

Parameter:	Definition:
Channel	the number of the RS-232 port. (The only valid value is 0).
Control	the address of a control file element used for the control status bits.
Characters	the number of characters in the buffer (including the end-of-line/termination characters) that the processor finds (0-256). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

The processor determines the number of characters (up to and including the end-of-line/termination characters) and puts this value in the position field. The done bit is then set. If a zero appears in the position field, no end-of-line/termination characters were found. The .FD bit is set if the position field was set to a non-zero value.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

The error bit (.ER) is set during the execution of the instruction if:

- the instruction is aborted – serial port not in User mode
- the instruction is aborted due to processor mode change

Number of Characters in Buffer (ACB)

Description:



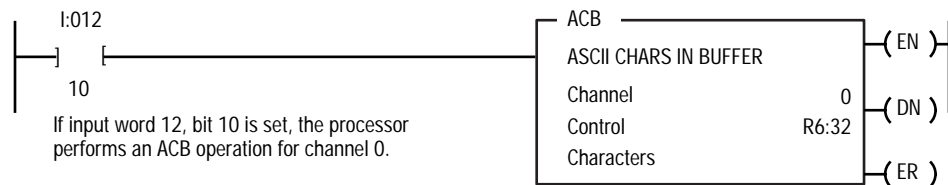
Use the ACB instruction to see how many total characters are in the buffer. On a false-to-true transition, the system determines the total number of characters and reports it in the Characters field. The serial port must be in User mode.

Entering Parameters

To use the ACB instruction, you must provide the processor the following information:

Parameter:	Definition:
Channel	the number of the RS-232 port (The only valid value in this field is 0).
Control	the address of a control file element used for control status bits.
Characters	the number of the characters in the buffer that the processor finds (0-256). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. When the instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

The processor determines the number of characters in the buffer and puts this value in the position field. The done bit is then set. If a zero appears in the position field, no characters were found. The .FD bit is set if the position field was set to a non-zero value.

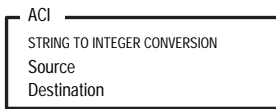
When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

The error bit (.ER) is set during the execution of the instruction if:

- the instruction is aborted – serial port not in User mode
- the instruction is aborted due to processor mode change

ASCII String to Integer (ACI)

Description:



Use the ACI instruction to convert an ASCII string to an integer value between $-32,768$ and $32,767$.

The processor searches the source (file type ST) for the first character that is between 0 and 9. All numeric characters are extracted until a non-numeric character or the end of the string is reached. Commas and signs ($-$, $+$) are allowed in the string.

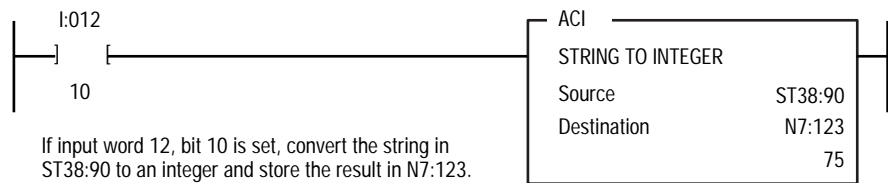
The extracted numeric string is then converted to an integer between $-32,768$ and $32,767$.

If no numeric characters are found, no action is taken. Also, if the string has an invalid length (less than zero or greater than 82), the fault bit is set (S:17/8) and the instruction is not executed.

This instruction also sets the arithmetic flags (found in word 0, bits 0-3 in the processor status file S):

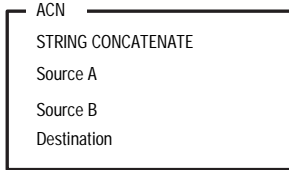
Bit:	Description:	Indicating:
S:0/0	Carry (C)	the carry was generated while converting the string to an integer
S:0/1	Overflow (V)	the integer value was outside of the valid range
S:0/2	Zero (Z)	the integer value is zero
S:0/3	Sign (S)	the integer value is negative

Example:



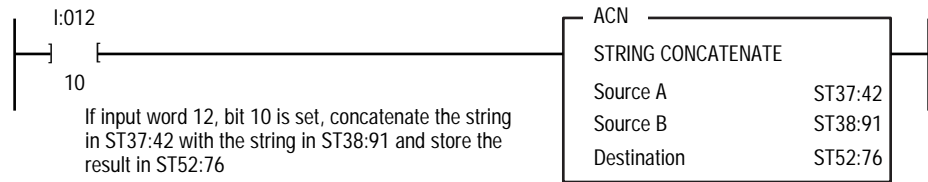
ASCII String Concatenate (ACN)

Description: The ACN instruction appends Source B to the end of Source A and stores the result in the Destination.



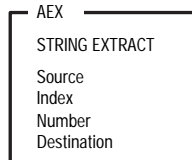
If the result is longer than 82 characters, only the first 82 are written to the destination file and the error bit (S:17/8) is set. Also, if the length of either string is invalid (less than zero or greater than 82), the fault bit is set and the string at the destination address is not changed.

Example:



ASCII String Extract (AEX)

Description: Use the AEX instruction to create a new string by taking a portion of an existing string.

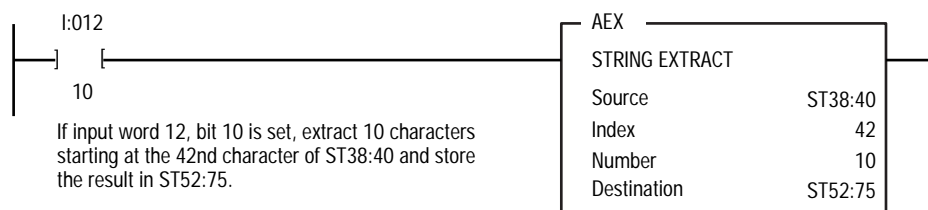


Entering Parameters

To use the AEX instruction, you must provide the processor the following information

Parameter:	Definition:
Source	the existing string.
Index	the starting position (from 1 to 82) of the portion of the string you want to extract. (An index of 1 indicates the left-most character of the string.)
Number	the number of characters (from 0 to 82) you want to extract, starting at the indexed position. If the Index plus the Number is greater than the total characters in the source string, the destination string will be the characters from the index to the end of the source string. If you enter 0 for the number, the destination string length is set to zero.
Destination	the string element (ST) where you want the extracted string stored.

Example:

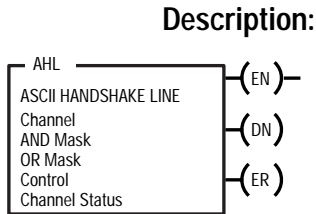


The following conditions cause the processor to set the fault bit (S:17/8):

- invalid string length or string length of zero
- index or number values outside of range
- index value greater than the length of the source string

The destination string will not change in any of the above instances.

ASCII Set or Reset Handshake Lines (AHL)



Use the AHL instruction to set or reset the RS-232 DTR and RTS handshake control lines for your modem. On a false-to-true transition, the system uses the two masks to determine whether to set or reset the DTR and RTS lines, or leave them unchanged.

Important: Before you use this instruction make sure that you do not conflict with the automatic control lines on your modem.

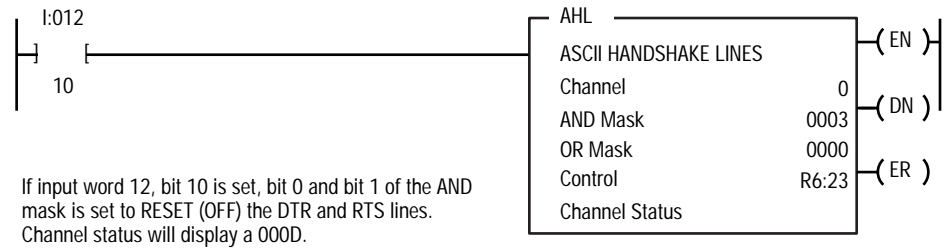
Entering Parameters

To use the AHL instruction, you must provide this information:

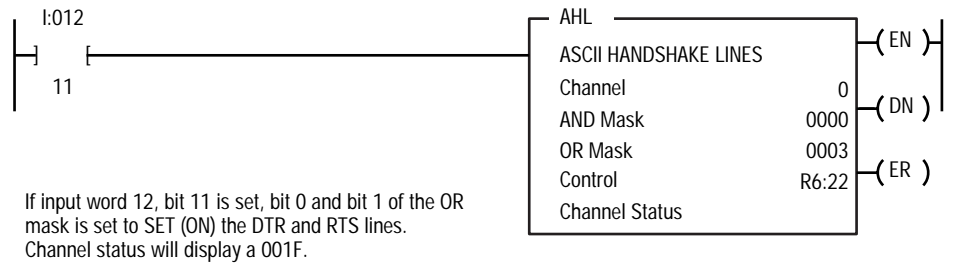
Parameter:	Definition:
Channel	the number of the RS-232 port that you want to use. Currently, only channel 0 can be set or reset.
AND Mask	the mask to reset the DTR and RTS control lines. Bit 0 corresponds to the DTR line and bit 1 corresponds to the RTS line. A 1 at the mask bit causes the line to be reset; a 0 leaves the line unchanged.
OR Mask	the mask to set the DTR and RTS control lines. Bit 0 corresponds to the DTR line and bit 1 corresponds to the RTS line. A 1 at the mask bit causes the line to be set; a 0 leaves the line unchanged.
Control	the address of the result control structure in the control area of memory for the result.
Channel Status	displays the current status (0000 to FFFF) of the handshake lines for the channel specified above. This field is display only; convert the hexadecimal status to binary and refer to the table below:

Bit	1	0
Line	RTS	DTR

Example: (Reset DTR and RTS Lines)



Example: (Set DTR and RTS Lines)

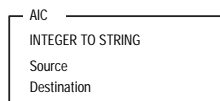


The error bit (.ER) is set during the execution of the instruction if the instruction is aborted due to processor mode change.

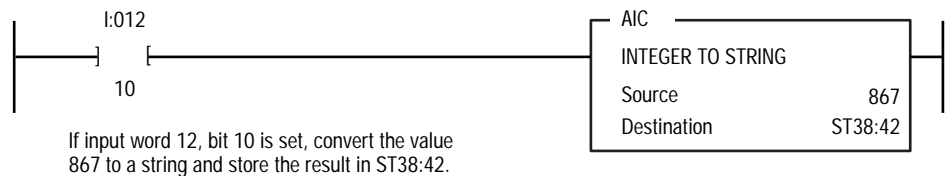
ASCII Integer to String (AIC)

Description:

Use the AIC instruction to convert an integer value (between -32,768 and 32,767) to an ASCII string. The source can be a constant or an integer address.

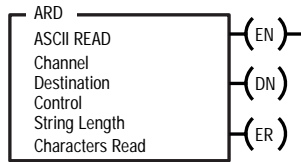


Example:



ASCII Read Characters (ARD)

Description:



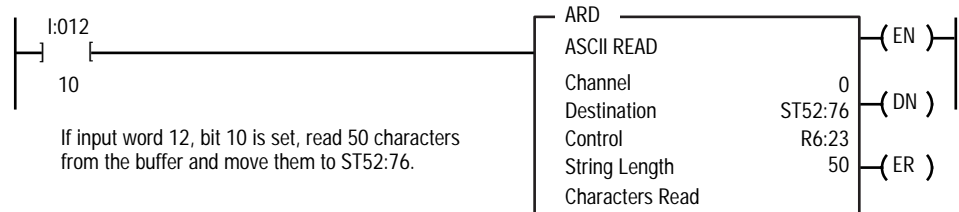
Use the ARD instruction to read characters from the buffer and store them in a string. To repeat the operation, the rung must go from false to true. The serial port must be in User mode.

Entering Parameters

To use the ARD instruction, provide the following information:

Parameter:	Definition:
Channel	the number of the RS-232 port. (The only valid value is 0).
Control	the control file element used for the control status bits.
Destination	the string element where you want the characters stored.
String Length	the number of characters you want to read from the buffer. The maximum is 82 characters. If you specify a length larger than 82, only 82 characters will be read. (If you specify 0, the string length defaults to 82.)
Characters Read	the number of characters that the processor moved from the buffer to the string (0 to 82). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Once the requested number of characters are in the buffer, the characters are moved to the destination string. The number of characters moved is put in the position word of the control element and the done bit is set.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

You can use the .UL bit to terminate an ARD instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port is not sending data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

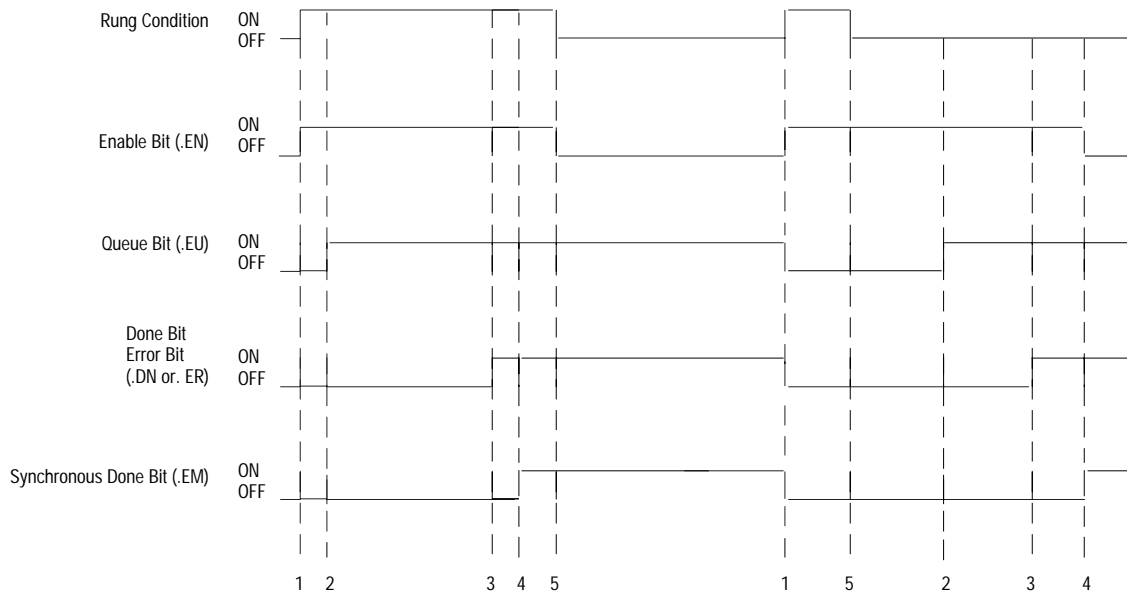
Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an ARD instruction starts executing with the .UL bit already set and there are **no** characters in the buffer, the instruction terminates. If an ARD instruction starts executing with the .UL bit already set and there **are** characters in the buffer, the instruction proceeds to normal completion.

The error bit (.ER) is set during the execution of the instruction if:

- the instruction is aborted – serial port not in User mode
- the instruction is aborted due to processor mode change
- when using a modem, the modem is disconnected

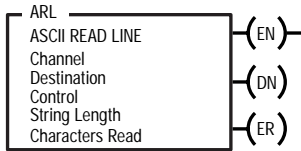
Figure 17.1
Example ARD Timing Diagram



- 1 - rung goes true
- 2 - instruction successfully queued
- 3 - instruction execution complete
- 4 - instruction scanned for the first time after execution is complete
- 5 - rung goes false

ASCII Read Line (ARL)

Description:



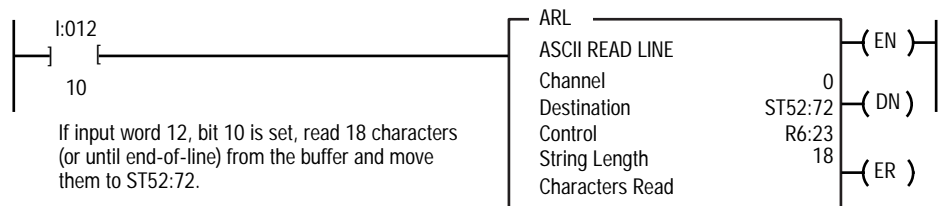
Use the ARL instruction to read characters from the buffer up to and including the end-of-line (termination) characters and store them in a string. The end-of-line characters are specified on the Channel Configuration screen (the default is a carriage return). For more information on channel configuration, see your software user manual. The serial port must be in User mode.

Entering Parameters

To use the ARL instruction, you must provide this information:

Parameter:	Definition:
Channel	the number of the RS-232 port. (The only valid value is 0).
Control	the address of the control file element used for the control status bits.
Destination	the string element where you want the string stored.
String Length	the number of characters (maximum 82) you want to read from the buffer. If the processor finds the end-of-line characters before reading the number of characters you specified, only those characters read and the end-of-line are moved to the destination.
Characters Read	the number of characters that the processor moved from the buffer to the string (0 to 82). This field is display only.

Example:



When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Once the requested number of characters (or the end-of-line characters) are in the buffer, all characters (including the end-of-line characters) are moved to the destination string. The number of characters moved is stored in the position word of the control element and the done bit is set.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

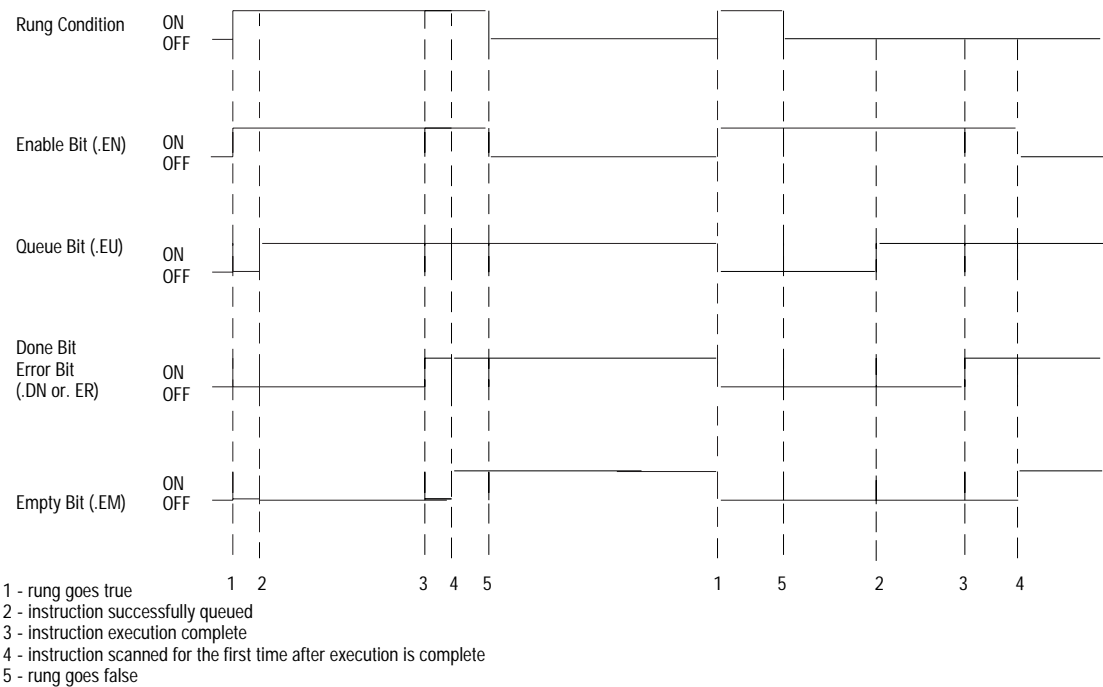
You can use the .UL bit to terminate an ARL instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port is not sending data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an ARL instruction starts executing with the .UL bit already set and there are **no** characters in the buffer, the instruction terminates. If an ARL instruction starts executing with the .UL bit already set and there **are** characters in the buffer, the instruction proceeds to normal completion.

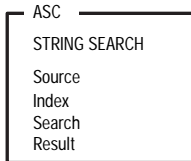
The error bit (.ER) is set during the execution of the instruction if the channel is in system mode (or switches to system mode), the processor switches to program/test mode or if the modem is lost (when using modem control).

Figure 17.2
Example ARL Timing Diagram



ASCII String Search (ASC)

Description: Use the ASC instruction to search an existing string (search string) for an occurrence of the source string.

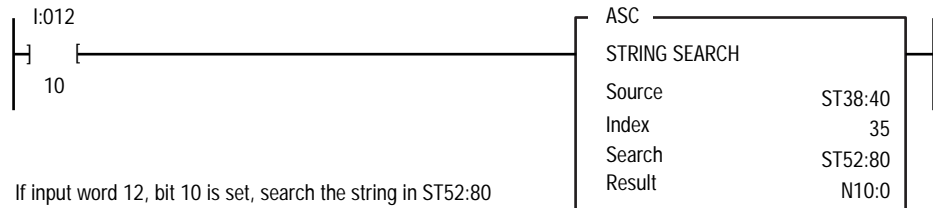


Entering Parameters

To use the ASC instruction, you must provide this information:

Parameter:	Definition:
Search	the string you want to examine.
Source	the string you want to find when examining the search string.
Index	the starting position (from 1 to 82) of the portion of the search string you want to search. An index of 1 indicates the left-most character.
Result	an integer address where the processor stores the position of the search string where the source string begins. If no match is found, 0 is stored in the result.

Example:



If input word 12, bit 10 is set, search the string in ST52:80 starting at the 35th character, for the string found in ST38:40. In this example, the result is stored in N10:0.

The following conditions cause the processor to set the fault bit (S:17/8):

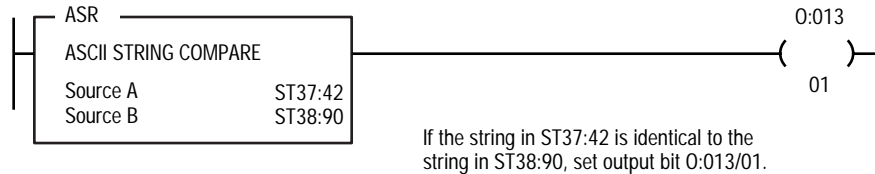
- invalid string length or string length of zero
- index values outside of range
- index value greater than the length of the source string

The result is set to zero in any of the above instances.

ASCII String Compare (ASR)

Description: Use the ASR instruction to compare two ASCII strings. The system looks for a match in length and upper/lower case. If the two strings are identical, the rung is true; if there are any differences, the rung is false.

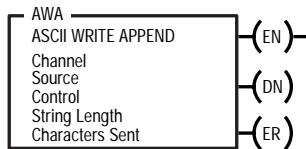
Example:



An invalid string length causes the processor to set the fault bit (S:17/8), and the rung is false.

ASCII Write with Append (AWA)

Description:

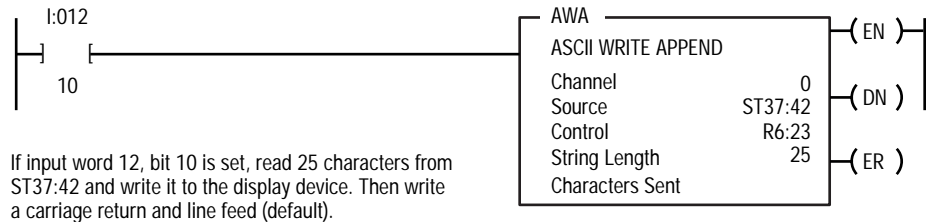


Use the AWA instruction to write characters from the source to a display device. This append instruction adds 1 or 2 characters (which you configure in the Channel Configuration). The default is a carriage return and line feed appended to the end of the string. You can use this instruction with the serial port in User or System mode.

Entering Parameters

To use the AWA instruction, you must provide this information:

Parameter:	Definition:
Channel	the number of the RS-232 port. (The only valid value is 0).
Source	the string you want to write.
Control	the address of the control file element used for the control status bits.
String Length	the maximum number of characters you want to write from the source string (0 to 82). If you enter 0, the entire string will be written.
Characters Sent	the number of characters that the processor sent to the display area (0 to 82). Only after the entire string is sent is this field updated (no running total for each character sent is stored). This field is display only.

Example:

When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Twenty-five characters from the start of string ST37:42 are sent to the display device and then user-configured append characters are sent. The done bit is set and a value of 27 is sent to the position word.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit to act as a secondary done bit corresponding to the program scan.

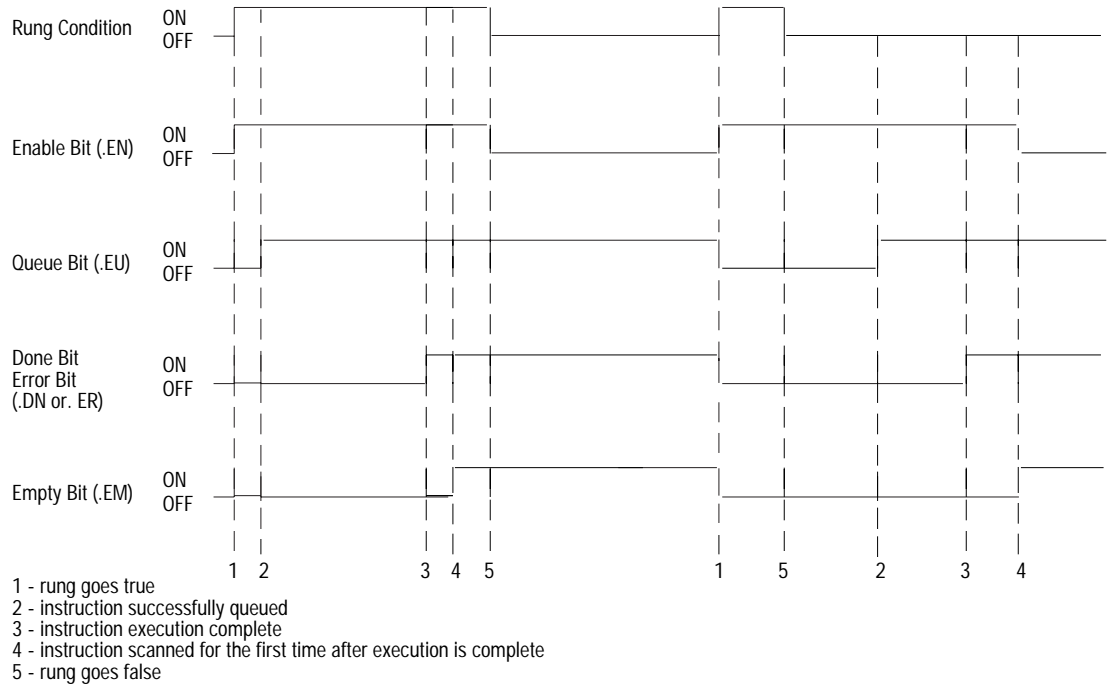
You can use the .UL bit to terminate an AWA instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port cannot accept data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an AWA instruction starts executing with the .UL bit already set, the instruction aborts immediately.

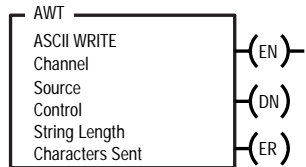
The error bit (.ER) is set during the execution of the instruction if the instruction is aborted due to processor mode change or if the modem becomes lost (when using modem control). If the modem was already lost, the instruction still executes.

Figure 17.3
Example AWA Timing Diagram



ASCII Write (AWT)

Description:

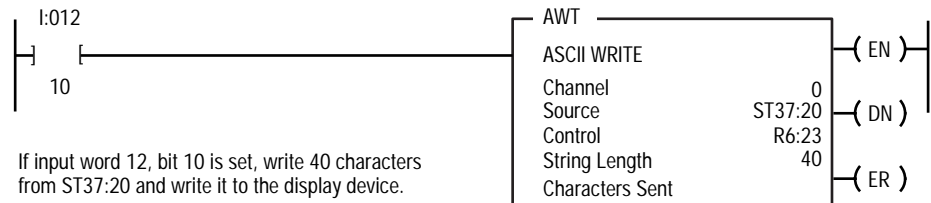


Use the AWT instruction to write characters from the source to a display device. To repeat the instruction, the rung must go from false to true. You can use this instruction with the port in System or User mode.

Entering Parameters

To use the AWT instruction, you must provide this information:

Parameters:	Definition:
Channel	the number of the RS-232 port. (The only valid value is 0).
Source	the string you want to write.
Control	the address of the control file element used for the control status file.
String Length	the maximum number of characters you want to write from the source string (0 to 82). If you enter 0, the entire string will be written.
Characters Sent	the number of characters that the processor sent to the display area (0 to 82). Only after the entire string is sent is this field updated (no running total for each character sent is stored). This field is display only.

Example:

When the rung goes from false to true, the control element enable bit (.EN) is set. The instruction is put in the ASCII instruction queue, the .EU bit is set and program scan continues. The instruction is then executed parallel to program scan.

Forty characters from string ST37:20 are sent through channel 0. The done bit is set and a value of 40 is sent to the position word.

When the program scans the instruction and finds the .DN bit set, the processor then sets the .EM bit. The .EM bit acts as a secondary done bit corresponding to the program scan.

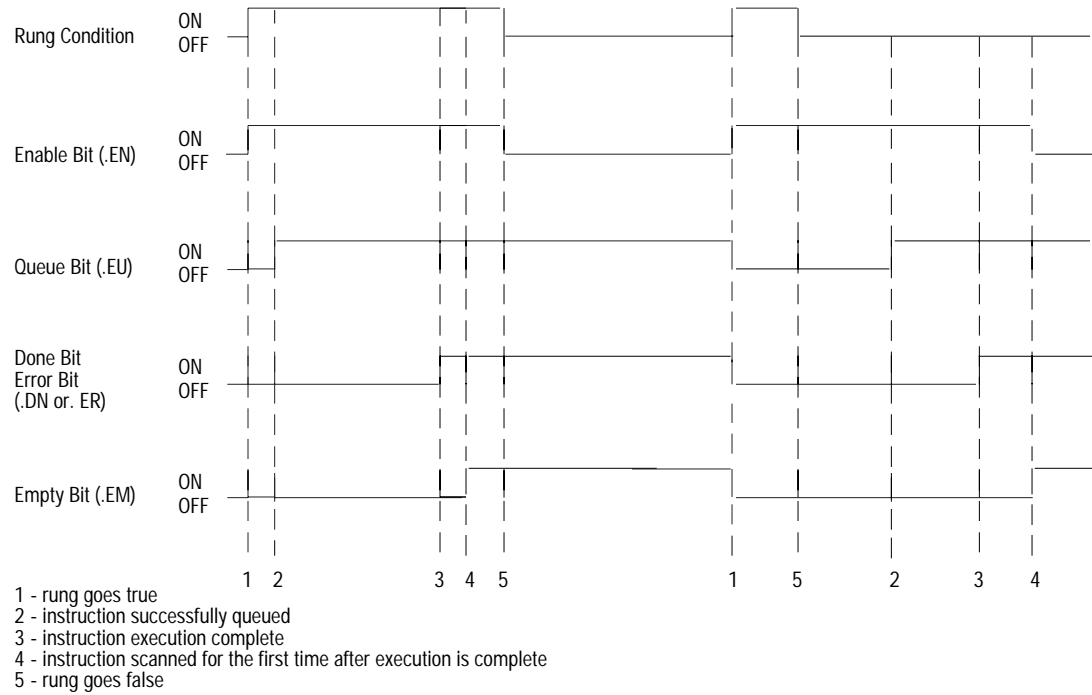
You can use the .UL bit to terminate an AWT instruction before it completes (for example, you may want to terminate the instruction if you know that the ASCII device connected to the port cannot accept data, or if the connection breaks after the instruction starts executing). Set the .UL bit in the control structure (the .ER bit is then set).

Important: When you set the .UL bit, the instruction does not terminate immediately; it may take several seconds.

If an AWT instruction starts executing with the .UL bit already set, the instruction aborts immediately.

The error bit (.ER) is set during the execution of the instruction if the processor switches to program or test mode or if the modem becomes lost (when using modem control). If the modem was already lost, the instruction still executes.

Figure 17.4
Example AWT Timing Diagram



Notes:

Custom Application Routine Instructions SDS, DFA

Chapter Objectives

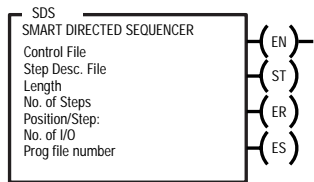
This chapter introduces the Custom Application Routine (CAR) instructions (SDS and DFA) for PLC-5 programming software. You need the Custom Application Routine (CAR) software in order to use these instructions.

For Information About: **See:**

SDS or DFA CAR utilities	Distributed Diagnostic and Machine Control User Manual
AGA3	PLC-5 AGA Mass Flow Custom Application Routine Programming Manual
AGA7	PLC-5 Volumetric Flow CARs for Turbine and Displacement Metering User Manual
NX19	PLC-5 Volumetric Flow CARs for Orifice Metering User Manual
API	PLC-5 Volumetric Flow CARs for Turbine and Displacement Metering User Manual

For more information on the operands (and valid data types/values of each operand) used by the instructions discussed in this chapter, see Appendix C.

Smart Directed Sequencer (SDS) Overview



The Smart Directed Sequencer (SDS) instruction provides state control that can be used to characterize normal and abnormal conditions.

The SDS instruction allows two basic types of logic equations:

- transitional
- combinatorial

**This Type of
Logic Equation:** **Does this:**

Transitional	provides traditional state-based control. This type of SDS instruction is built around the state transition concept, where each input transition directs the instruction to a unique next state using a logical OR structure. One input change directs the instruction to step A, another to step B, etc.
--------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Combinatorial	provides for the ANDing of inputs in addition to the OR function used in transition equations. This allows complex combinations to be accommodated more easily within the SDS framework with a minimum number of steps.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Programming the SDS Instruction

To program the SDS instruction, you have to:

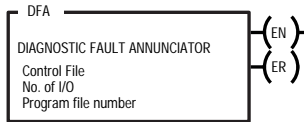
- download the SDS CAR
- enter the SDS instruction
- enter the configuration information
- enter I/O information

Important: You cannot use the BT, PD, MG, ST, or SC data types within the I/O list of the SDS instruction.

Important: When you enter the `Control File` and `Step Desc. File` operands, make certain the file numbers (i.e., 7, 10) are not the same.

For more information on the SDS instruction, see the Distributed Diagnostic and Machine Control User Manual.

Diagnostic Fault Annunciator (DFA) Overview



The Diagnostic Fault Annunciator (DFA) instruction monitors inputs you define, but it cannot control outputs. Valid inputs can be:

- storage points, such as binary bits
- counter / timer done bits
- outputs (real or logical)
- any valid bit address
- lube level indicators
- alarms
- fault bits set by another device (like an IMC motion controller) or by ladder logic

You can use the DFA instruction to generate messages when a fault occurs. In addition, you can create other types of operational and diagnostic messages with the DFA instruction, such as tool change messages and operating instructions.

Programming the DFA Instruction

To program the DFA instruction, you have to:

- download the DFA CAR
- enter the DFA instruction
- enter the configuration information
- enter I/O information

For more information on the DFA instruction, see the Distributed Diagnostic and Machine Control User Manual.

Notes:

Instruction Timing and Memory Requirements

Instruction Timing and Memory Requirements

The time it takes for a processor to scan an instruction depends on the type of instruction, the type of addressing, the type of data, whether the instruction has to convert data, and whether the instruction is true or false.

The timing and memory requirements estimates in this chapter have the following assumptions:

- direct addressing
- integer data, except where noted
- no data-type conversions
- addresses within first 4098 words of the data table for Classic PLC-5 processors; addresses within first 2048 words for Enhanced PLC-5 processors
- execution times shown in μ s

Memory requirements refer to the number of words the instruction uses. In some cases, an instruction may have a range of memory requirements. The range of words exists because the instruction can use different types of data and addressing modes.

The tables are divided into instruction times and memory requirements that are specific to each processor.

If You Are Using this Processor:	See Page:
Enhanced PLC-5, series C	
Bit and Word Instructions	A-2
File Instructions	A-5
Classic PLC-5 (all series):	
Bit and Word Instructions	A-10
File Instructions	A-13

Timing for Enhanced PLC-5 Processors

Bit and Word Instructions

Table A.A shows timing and memory requirements for bit and word instructions for Enhanced PLC-5 processors.

Table A.A
Timing and Memory Requirements for Bit and Word Instructions
(Enhanced PLC-5 Processors)

Category	Code	Title	Execution Time (μs) integer		Execution Time (μs) floating point		Words of Memory ¹
			True	False	True	False	
Relay	XIC	examine if closed	.32	.16			1 ²
	XIO	examine if open	.32	.16			1 ²
	OTL	output latch	.48	.16			1 ²
	OTU	output unlatch	.48	.16			1 ²
	OPE	output energize	.48	.48			1 ²
Branch		branch end	.16	.16			1
		next branch	.16	.16			1
		branch start	.16	.16			1
Timer and Counter	TON	timer on (0.01 base) (1.0 base)	3.8	2.6			2-3
			4.1	2.5			
	TOF	timer off (0.01 base) (1.0 base)	2.6	3.2			2-3
			2.6	3.2			
	RTO	retentive timer on (0.01 base) (1.0 base)	3.8	2.4			2-3
			4.1	2.3			
CTU	count up	3.4	3.4			2-3	
CTD	count down	3.3	3.4			2-3	
RES	reset	2.2	1.0			2-3	

(Continued)

1. Use the larger number for addresses beyond 2048 words in the processor's data table.

2. For every bit address above the first 256 words of memory in the data table, add 0.16 ms and 1 word of memory.

Category	Code	Title	Execution Time (μ s) integer		Execution Time (μ s) floating point		Words of Memory ¹
			True	False	True	False	
Arithmetic	ADD	add	6.1	1.4	14.9	1.4	4-7
	SUB	subtract	6.2	1.4	15.6	1.4	4-7
	MUL	multiply	9.9	1.4	18.2	1.4	4-7
	DIV	divides	12.2	1.4	23.4	1.4	4-7
	SQR	square root	9.9	1.3	35.6	1.3	3-5
	NEG	negate	4.8	1.3	6.0	1.3	3-5
	CLR	clear	3.4	1.1	3.9	1.1	2-3
	AVE	average file	152+E25.8	30	162+E22.9	36	4-7
	STD	standard deviation	321+E84.3	34	329+E77.5	34	4-7
	TOD	convert to BCD	7.8	1.3			3-5
	FRD	convert from BCD	8.1	1.3			3-5
	RAD	radian	57.4	1.4	50.1	1.4	3-5
	DEG	degree	55.9	1.4	50.7	1.4	3-5
	SIN	sine			414	1.4	3-5
	COS	cosine			404	1.4	3-5
	TAN	tangent			504	1.4	3-5
	ASN	inverse sine			426	1.4	3-5
	ACS	inverse cosine			436	1.4	3-5
	ATN	inverse tangent			375	1.4	3-5
	LN	natural log	409	1.4	403	1.4	3-5
	LOG	log	411	1.4	403	1.4	3-5
	XPY	X to the power of Y	897	1.5	897	1.5	4-7
	SRT	sort file					3-5
	(5/11, -5/20)	276 + 12[E**1.34]	227	278 + 16[E**1.35]	227		
	(-5/30, -5/40, -5/60, -5/80)	224 + 25[E**1.34]	189	230 + 33[E**1.35]	189		

(Continued)

1. Use the larger number for addresses beyond 2048 words in the processor's data table.
E = number of elements acted on per scan
SRT true is only an approximation. Actual time depends on the randomness of the numbers.

Category	Code	Title	Execution Time (μ s) integer		Execution Time (μ s) floating point		Words of Memory ¹
			True	False	True	False	
Logic	AND	and	5.9	1.4			4-7
	OR	or	5.9	1.4			4-7
	XOR	exclusive or	5.9	1.4			4-7
	NOT	not	4.6	1.3			3-5
Move	MOV	move	4.5	1.3	5.6	1.3	3-5
	MVM	masked move	6.2	1.4			4-7
	BTD	bit distributor	10.0	1.7			6-9
Comparison	EQU	equal	3.8	1.0	4.6	1.0	3-5
	NEQ	not equal	3.8	1.0	4.5	1.0	3-5
	LES	less than	4.0	1.0	5.1	1.0	3-5
	LEQ	less than or equal	4.0	1.0	5.1	1.0	3-5
	GRT	greater than	4.0	1.0	5.1	1.0	3-5
	GEQ	greater than or equal	4.0	1.0	5.1	1.0	3-5
	LIM	limit test	6.1	1.1	8.4	1.1	4-7
	MEQ	mask compare if equal	5.1	1.1			4-7
Compare	CMP	all	$2.48 + (\Sigma[0.8 + i])$	$2.16 + Wi[0.56]$	$2.48 + (\Sigma[0.8 + i])$	$2.16 + Wi[0.56]$	$2+Wi$
Compute	CPT	all	$2.48 + (\Sigma[0.8 + i])$	$2.16 + Wi[0.56]$	$2.48 + (\Sigma[0.8 + i])$	$2.16 + Wi[0.56]$	$2+Wi$

1. Use the larger number for addresses beyond 2048 words in the processor's data table.

i = execution time of each instruction (operation, e.g. ADD, SUB, etc.) used within the CMP or the CPT expression

Wi = number of words of memory used by the instruction (operation, e.g. ADD, SUB, etc.) within the CMP or CPT expression

CMP or CPT instructions are calculated with short direct addressing

File Instructions

Refer to Table A.B for the instruction timing for file instructions.

Table A.B
Timing and Memory Requirements for File, Program Control, and ASCII
Instructions (Enhanced PLC-5 Processors)

Category	Code	Title	Time (μs)		Time (μs)		Words of Memory ¹
			integer True	False	floating point True	False	
File Arithmetic and Logic	FAL	all	$11 + (\Sigma[2.3 + i])E$	$6.16 + Wi[0.16]$	$11 + (\Sigma[2.3 + i])E$	$6.16 + Wi[0.16]$	$3-5 + Wi$
File Search and Compare	FSC	all	$11 + (\Sigma[2.3 + i])E$	$6.16 + Wi[0.16]$	$11 + (\Sigma[2.3 + i])E$	$6.16 + Wi[0.16]$	$3-5 + Wi$
File	COP	copy	$16.2 + E[0.72]$	1.4	$17.8 + E[1.44]$	1.4	4-6
		counter, timer, and control	$15.7 + E[2.16]$	1.4			
	FLL	fill	$15.7 + E[0.64]$	1.5	$18.1 + E[0.80]$	1.5	4-6
		counter, timer, and control	$15.1 + E[1.60]$	1.5			
Shift Register	BSL	bit shift left	$10.6 + B[0.025]$	5.2			4-7
	BSR	bit shift right	$11.1 + B[0.025]$	5.2			4-7
	FFL	FIFO load	8.9	3.8			4-7
	FFU	FIFO unload	$10.0 + E[0.43]$	3.8			4-7
	LFL	LIFO load	9.1	3.7			4-7
	LFU	LIFO unload	10.6	3.8			4-7
Diagnostic	FBC	0 mismatch	$15.4 + B[0.055]$	2.9			6-11
		1 mismatch	$22.4 + B[0.055]$	2.9			
		2 mismatches	$29.9 + B[0.055]$	2.9			
	DDT	0 mismatch	$15.4 + B[0.055]$	2.9			6-11
		1 mismatch	$24.5 + B[0.055]$	2.9			
		2 mismatches	$34.2 + B[0.055]$	2.9			
	DTR	data transitional	5.3	5.3			4-7

(Continued)

1. Use the larger number for addresses beyond 2048 words in the processor's data table.

i = execution time of each instruction (operation, e.g. ADD, SUB, etc.) used within the FAL or the FSC expression

E = number of elements acted on per scan

B = number of bits acted on per scan

Wi = number of words of memory used by the instruction (operation, e.g. ADD, SUB, etc.) within the FAL or FSC expression

FAL or FSC instructions are calculated with short direct addressing

Category	Code	Title	Time (μs)		Time (μs)		Words of Memory ¹	
			integer True	False	floating point True	False		
Sequencer	SQL	sequencer input	7.9	1.3			5-9	
	SQL	sequencer load	7.9	3.5			4-7	
	SQO	sequencer output	9.7	3.7			5-9	
Immediate I/O ²	IIN	immediate input		1.1			2	
		(-5/11, -5/20) (-5/30, -5/40, -5/60, -5/80)	357 307					
	IOT	immediate output		1.1			2	
		(-5/11, -5/20) (-5/30, -5/40, -5/60, -5/80)	361 301					
	IDI	immediate data input		1.1			4-7	
(-5/20C) (-5/40C, -5/60C, and -5/80C)		200 + 1.4 (for each word) 200 + 1.4 (for each word)						
IDO	immediate data output		1.1			4-7		
	(-5/20C) - (-5/40C, -5/60C, and -5/80C)	230 + 1.4 (for each word) - 250 + 1.7 (for each word)						
Zone Control	MCR	master control	0.16	0.16			1	
Program Control	JMP	jump	8.9 + (file number - 2) * 0.96	1.4 + (file number - 2) * 0.96			2	
	LBL	label	0.32	0.32			2+position in label table	
	JSR ³ / RET	jump to subroutine/ return PLC-5/11, -5/20, -5/30, -5/40, -5/40L, -5/60, -5/60L, -5/20E, -5/40E	- 0 parameters	12.3	1.0	n/a	n/a	3+ parameters/ JSR 1+ parameters/ RET
			- 1 parameter	16.1	1.0	17.3	1.0	
			- increase/parameter	3.8	n/a	5.0	n/a	
			PLC-5/80					
			- 0 parameters	315	1.0			
	- 1 parameter	340	1.0	349				
	- increase/parameter	31	n/a	33	1.0			

(Continued)

1. Use the larger number for addresses beyond 2048 words in the processor's data table.

2. Timing for immediate I/O instructions is the time for the instruction to queue-up for processing

3. Calculate execution times as follows: (time) + (quantity of additional parameters)(time/parameter). For example, if you are passing 3 integer parameters in a JSR within a PLC-5/11 processor, the execution time = 16.1 + (2)(3.8)=23.7μs.

B = number of bits acted on per scan

Category	Code	Title	Time (μs) integer		Time (μs) floating point		Words of Memory ¹
			True	False	True	False	
Program Control	SBR	0 parameters	12.3	1.0			1+ parameters
		1 parameter	16.1	1.0	17.3	1.0	
		increase/ parameter	3.8		5.0		
	END	end	negligible				1
	TND	temporary end					1
	EOT	end of transition					1
	AFI	always false	0.16	0.16			1
	ONS	one shot	3.0	3.0			2-3
	OSR	one shot rising	6.2	6.0			4-6
	OSF	one shot falling	6.2	5.8			4-6
	FOR/ NXT	for next loop (PLC-5/80)	8.1 + L[15.9] (151+L[277])	5.3 + N[0.75] (152+N[6.1])			FOR 5-9 NXT 2
	BRK	break	11.3 + N[0.75]	0.9			1
	UID	user interrupt disable (-5/11, -5/20) (-5/30, -5/40, -5/60, -5/80)	175 119	1.0			1
	UIE	user interrupt enable (-5/11, -5/20) (-5/30, -5/40, -5/60, -5/80)	170 100	1.0			1

(Continued)

1. Use the larger number for addresses beyond 2048 words in the processor's data table.

L = number of FOR/NXT loops

N = number of words in memory between FOR/NXT or BRK/NXT

Category	Code	Title	Time (μs)		Time (μs)		Words of Memory ¹
			integer True	False	floating point True	False	
Process Control	PID	PID loop control					5-9
	Gains	Independent (-5/11, -5/20, -5/20E, -5/20C)	462	3.0	882	58	
		(-5/30, -5/40, -5/40E, -5/40C, -5/40L, -5/60, -5/60C, -5/60L, -5/80, -5/80E, -5/80C)	655				
		ISA (-5/11, -5/20, -5/20E, -5/20C)	560		1142		
		(-5/30, -5/40, -5/40E, -5/40C, -5/40L, -5/60, -5/60C, -5/60L, -5/80, -5/80E, -5/80C)	895				
	Modes	Manual (-5/11, -5/20, -5/20E, -5/20C)	372		900		
		(-5/30, -5/40, -5/40E, -5/40C, -5/40L, -5/60, -5/60C, -5/60L, -5/80, -5/80E, -5/80C)	420				
		Set Output (-5/11, -5/20, -5/20E, -5/20C)	380		882		
		(-5/30, -5/40, -5/40E, -5/40C, -5/40L, -5/60, -5/60C, -5/60L, -5/80, -5/80E, -5/80C)	440				
	Cascade	Slave			1286		
		Master			840		
ASCII ²	ABL ²	test buffer for line (-5/11, -5/20)	316	214			3-5
		(-5/30, -5/40, -5/60, -5/80)	388	150			
	ACB ²	no. of characters in buffer (-5/11, -5/20)	316	214			3-5
		(-5/30, -5/40, -5/60, -5/80)	389	150			
	ACI	string to integer (-5/11, -5/20)	220 + C[11]	1.4			3-5
		(-5/30, -5/40, -5/60, -5/80)	140 + C[21.4]				

(Continued)

1. Use the larger number for addresses beyond 2048 words in the processor's data table.
 2. Timing for ASCII instructions is the time for the instruction to queue-up for processing in channel 0.

Category	Code	Title	Time (μs) integer		Time (μs) floating point		Words of Memory ¹
			True	False	True	False	
ASCII ²	ACN	string concatenate		1.9			4-7
		(-5/11, -5/20)	237 + C[2.6]				
		(-5/30, -5/40, -5/60, -5/80)	179 + C[5.5]				
	AEX	string extract		1.9			5-9
		(-5/11, -5/20)	226 + C[1.1]				
		(-5/30, -5/40, -5/60, -5/80)	159 + C[2.2]				
	AHL ²	set or reset lines					5-9
		(-5/11, -5/20)	318	213			
		(-5/30, -5/40, -5/60, -5/80)	526	157			
	AIC	integer to string		1.4			3-5
		(-5/11, -5/20)	260				
		(-5/30, -5/40, -5/60, -5/80)	270				
ARD ²	read characters					4-7	
	(-5/11, -5/20)	315	214				
	(-5/30, -5/40, -5/60, -5/80)	380	149				
ARL ²	read line					4-7	
	(-5/11, -5/20)	316	214				
	(-5/30, -5/40, -5/60, -5/80)	388	151				
ASC	string search		1.9			5-9	
	(-5/11, -5/20)	222 + C[1.7]					
	(-5/30, -5/40, -5/60, -5/80)	151 + C[3.0]					
ASR	string compare					3-5	
	(-5/11, -5/20)	234 + C[1.3]	202				
	(-5/30, -5/40, -5/60, -5/80)	169 + C[2.4]	119				
AWA ²	write with append					4-7	
	(-5/11, -5/20)	319	215				
	(-5/30, -5/40, -5/60, -5/80)	345	154				
AWT ²	write					4-7	
	(-5/11, -5/20)	318	215				
	(-5/30, -5/40, -5/60, -5/80)	344	151				

1. Use the larger number for addresses beyond 2048 words in the processor's data table.
 2. Timing for ASCII instructions is the time for the instruction to queue-up for processing in channel 0.
 C = number of ASCII characters

Timing for Classic PLC-5 Processors

Bit and Word Instructions

Table A.C shows timing and memory requirements for bit and word instructions for Classic PLC-5 processors.

Table A.C
Timing and Memory Requirements for Bit and Word Instructions
(Classic PLC-5 Processors)

Category	Code	Title	Execution Time (μ s) integer		Execution Time (μ s) floating point		Words of Memory ¹	
			True	False	True	False		
Relay	XIC	examine if closed	1.3	0.8			1 ²	
	XIO	examine if open	1.3	0.8			1 ²	
	OTL	output latch	1.6	0.8			1 ²	
	OTU	output unlatch	1.6	0.8			11	
	OTE	output energize	1.6	1.6			1 ²	
Branch		branch end	0.8	0.8			1	
		next branch	0.8	0.8			1	
		branch start	0.8	0.8			1	
Timer and Counter	TON	timer on	(0.01 base)	39	27			2-3
			(1.0 base)	44	28			
	TOF	timer off	(0.01 base)	30	43			2-3
			(1.0 base)	30	51			
	RTO	retentive timer on	(0.01 base)	39	24			2-3
			(1.0 base)	44	24			
	CTU	count up		32	34			2-3
CTD	count down		34	34			2-3	
RES	reset		30	14			2-3	

¹ Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

² For every bit address above the first 256 words of memory in the data table, add 0.8 μ s to the execution time and 1 word of memory to the requirements.

(Continued)

Category	Code	Title	Execution Time (μ s) integer		Execution Time (μ s) floating point		Words of Memory ¹
			True	False	True	False	
Arithmetic	ADD	add	36	14	92	14	4-7
	SUB	subtract	36	14	92	14	4-7
	MUL	multiply	41	14	98	14	4-7
	DIV	divide	49	14	172	14	4-7
	SQR	square root	82	14	212	14	3-5
	NEG	negate	28	14	36	14	3-5
	CLE	clear	18	14	23	14	2-3
	TOD	convert to BCD	52	14			3-5
	FRD	convert from BCD	44	14			3-5
Logic	AND	and	36	14			4-7
	OR	or	36	14			4-7
	XOR	exclusive or	36	14			4-7
	NOT	not	27	14			3-5
Move	MOV	move	26	14	35	14	3-5
	MVM	masked move	55	14			6-9
Comparison	EQU	equal	32	14	42	14	3-5
	NEQ	not equal	32	14	42	14	3-5
	LES	less than	32	14	42	14	3-5
	LEQ	less than or equal	32	14	42	14	3-5
	GRT	greater than	32	14	42	14	3-5
	GEQ	greater than or equal	32	14	42	14	3-5
	LIM	limit test	42	14	60	14	4-7
	MEQ	mask compare if equal	41	14			4-7

¹ Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

(Continued)

Category	Code	Title	Execution Time (μ s) integer		Execution Time (μ s) floating point		Words of Memory ¹
			True	False	True	False	
Compute	CPT	add	67	34	124	34	6-9
		subtract	67	34	124	34	6-9
		multiply	73	34	130	34	6-9
		divide	80	34	204	34	6-9
		square root	113	33	244	34	5-7
		negate	59	33	68	34	5-7
		clear	49	30	55	34	4-5
		move	58	33			5-7
		convert to BCD	84	33			5-7
		convert from BCD	75	33			5-7
		AND	68	34			6-9
		OR	68	34			6-9
		XOR	68	34			6-9
		NOT	59	34			5-7
Compare	CMP	equal	63	34	73	34	5-7
		not equal	63	34	73	34	5-7
		less than	63	34	73	34	5-7
		less than or equal	63	34	73	34	5-7
		greater than	63	34	73	34	5-7
		greater than or equal	63	34	73	34	5-7

¹Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

File Instructions

The instruction timing for file instructions depends on the data type, number of files acted on per scan, number of elements acted on per scan, and whether the instruction converts data between integer and floating point formats.

- for integer to floating point conversion, add:
 8 μ s for each element address
 10 μ s for each file address (# prefix)
- for floating point to integer conversion add:
 33 μ s for each element address
 44 μ s for each file address (# prefix)

Table A.D
Timing and Memory Requirements for File Instructions
(Classic PLC-5 Processors)

Category	Code	Title	Time (μ s)	Time (μ s)	Time (μ s)	Words of Memory ¹
			integer	floating point	integer or floating point	
			True	True	False	
File Arithmetic and Logic	FAL	add	$98 + W[36.7 + N]$	$98 + W[95.1 + N]$	54	7-12
		subtract	$98 + W[36.7 + N]$	$98 + W[95.1 + N]$	54	7-12
		multiply	$98 + W[42.5 + N]$	$98 + W[101.2 + N]$	54	7-12
		divide	$98 + W[51.1 + N]$	$98 + W[180.3 + N]$	54	7-12
		square root	$98 + W[84.7 + N]$	$98 + W[220.5 + N]$	54	6-10
		negate	$98 + W[29.2 + N]$	$98 + W[37.2 + N]$	54	6-10
		clear	$98 + W[18.4 + N]$	$98 + W[24.0 + N]$	54	5-8
		move	$98 + W[27.3 + N]$	$98 + W[36.2 + N]$	54	6-10
		convert to BCD	$98 + W[54.3 + N]$		54	6-10
		convert from BCD	$98 + W[45.4 + N]$		54	6-10

¹ Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.
 W = number of elements acted on per scan
 N = $2 \times$ (number of integer file addresses) + $8 \times$ (number of floating-point file addresses) + $6 \times$ (number of timer, counter, or control file addresses) + (number of conversions between integer and floating point formats)

(Continued)

Category	Code	Title	Time (μ s) integer	Time (μ s) floating point	Time (μ s) integer or floating point	Words of Memory ¹	
			True	True	False		
File Arithmetic and Logic		AND	$98 + W[37.2 + N]$		54	7-12	
		OR	$98 + W[37.2 + N]$		54	7-12	
		XOR	$98 + W[37.2 + N]$		54	7-12	
		NOT	$98 + W[28.2 + N]$		54	6-10	
File Search and Compare	FSC	all comparisons	$93 + W[32.7 + N]$	$93 + W[43.3 + N]$	54	6-10	
File	COP	copy	$88 + 2.7W$	$104 + 3.8W$	20	4-7	
		counter, timer, and control	$98 + 5.8W$				
	FLL	fill	$81 + 2.1W$	$100 + 3.1W$	15	4-7	
		counter, timer, and control	$97 + 4.4W$				
Shift Register	BSL	bit shift left	$74 + 3.4W$		57	4-7	
	BSR	bit shift right	$78 + 3.0W$		57	4-7	
	FFL	FIFO load	54		44	4-7	
	FFU	FIFO unload	$68 + 3.2W$		46	4-7	
Diagnostic	FBC	file bit compare				6-11	
		0 mismatch	$75 + 6W$		31		
		1 mismatch	$130 + 6W$		31		
		2 mismatches	$151 + 6W$		31		
	DDT	diagnostic detect					6-11
		0 mismatch	$71 + 6W$		31		
		1 mismatch	$150 + 6W$		31		
		2 mismatches	$161 + 6W$				

¹ Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

W = number of elements acted on per scan

N = $2 \times$ (number of integer file addresses) + $8 \times$ (number of floating-point file addresses) + $6 \times$ (number of timer, counter, or control file addresses) + (number of conversions between integer and floating point formats)

(Continued)

Category	Code	Title	Time (μ s) integer	Time (μ s) floating point	Time (μ s) integer or floating point	Words of Memory ¹	
			True	True	False		
Zone Control	MCR	master control	12		18	1	
Immediate I/O	IIN	immediate input				2-3	
		local	196		16		
		remote	204		16		
	IOT	immediate output				2-3	
		local	202		16		
		remote	166		16		
Sequencer	SQL	sequencer input	57		14	5-9	
	SQL	sequencer load	55		42	4-7	
	SQO	sequencer output	77		42	5-9	
Jump and Subroutine	JMP	jump	45		15	2-3	
	JSR	jump to subroutine					
	SBR	0 parameters	56		15	2-3	
		1 parameter	91		15	3-5	
		add per parameter	21				
	RET	return from sub.					
		0 parameters	48		13	1	
		1 parameter	70		13	2-3	
		add per parameter	21				
	LBL	label	12		12	3	

¹Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

(Continued)

Category	Code	Title	Time (μ s) integer	Time (μ s) floating point	Time (μ s) integer or floating point	Words of Memory ¹
			True	True	False	
Miscellaneous	END	end	negligible		negligible	1
	TND	temporary end	negligible		15	1
	AFI	always false	15		13	1
	ONS	one shot	28		30	2-3
	DTR	data transitional	41		41	4-7
	BTD	bit distributor	77		14	6-11
	PID	PID loop control	608		34	5-9
	BTR	block transfer read		See chapter 15		
	BTW	block transfer write				
	MSG	message		See chapter 16		

¹Use the smaller number if all addresses are below word 4096; use the larger number if all addresses are above 4096.

Program Constants

Use program constants in compare, compute, and file instructions to improve instruction execution times. Integer constants and floating-point constants execute in less than 1 μ s.

Direct and Indirect Elements – Enhanced PLC-5 Processors

Additional execution time for directly and indirectly addressed elements depends on location in memory, reference to the beginning of all data files (output file, word 0), whether data is stored at the source or destination address, and whether the instruction converts data. Table A.E lists times to add to instruction execution times.

Table A.E
Additional Execution Time (Enhanced PLC-5 Processors)

Addressing Mode	Data Type	Modifier in μ sec (add for each operand)
Direct	Integer	0
	Float	0
Index	Integer	1.1
	Float	1.8
	Counter-Timer-Control	2.4
Immediate	Integer	0.24
	Float	1.0
Indirect		$6.6 + W[0.09]$
Float-to-integer		5.6
Integer-to-float		8.4

Direct and Indirect Elements – Classic PLC-5 Processors

Additional execution time for directly addressed elements depends on location in memory, reference to the beginning of all data files (output file, word 0), whether data is stored at the source or destination address, and whether the instruction converts data. Table A.F lists times to add to instruction execution times.

Table A.F
Additional Execution Time Based on Source and Destination Addresses
(Classic PLC-5 Processors)

Data Type	Source (integer to floating point)			Destination (floating point to integer)		
	0-2K	2-4K	4K+	0-2K	2-4K	4K+
integer	0	1	2	0	1	2
floating point	0	3	4	0	3	4
data conversion	8	9	10	33	34	35

When file addresses (# prefix) in the expression or destination address contain indirect addresses for file numbers, add:

- 45 μ s when the indirect address is integer type
- 48 μ s when the indirect address is floating point type
- 48 μ s when the indirect address is timer, counter, or control type

When file addresses in the expression or destination contain indirect addresses for element numbers, add:

- 45 μ s when the indirect address is integer type
- 46 μ s when the indirect address is floating point type
- 46 μ s when the indirect address is timer, counter, or control type

If the file address contains two indirect addresses, add only one value (the largest). For example, for #F[N7:20][N7:30], add 48 μ s (indirect floating point file address).

Multiply the additional time by the number of elements in the file for any type of file or file address. For example:

Expression:#N[N7:100]:10 * F8:20
 add 10 for converting to floating point
 add 45 for indirect address

Destination:#N7:30
 add 44 for converting to integer

FAL multiply:98 + W[42.5 + N + indirect addressing]
 N = 2(2) + 8 (1) + 6(0) + 10 + 44 = 66
 W = 16

Execution time in ALL mode:
 98 + 16[42.5 + 66 + 45]
 2554 μs

**Indirect Bit or Elements Addresses
 – Classic PLC-5 Processors**

Additional execution times for indirectly addressed bits and elements depends on the number of indirect addresses in the overall address. Table A.G lists the additional times.

**Table A.G
 Additional Execution Times for Indirectly Addressed Bits and Elements
 Classic PLC-5 Processors**

Data Type	Time (μs) for Variable File or Element	Time (μs) for Variable File and Element
Bit in binary file	57	60
Bit in integer file	60	63
Bit in timer, counter, or control file	64	66
Integer (N)	42	42
Timer (T), counter (C), or control (R) file	43	44
Floating point (F)	61	64
Converting integer to floating point	71	77
Converting timer, counter, or control to floating point	85	81

Additional Timing Considerations – Classic PLC-5 Processors

Table A.H lists additional timing considerations.

Table A.H
Additional Timing Considerations (Classic PLC-5 Processors)

Tasks	Time (milliseconds)
Housekeeping	4.5 max
Resident Local I/O scan	1 per assigned rack number
Remote I/O scan	10 per assigned rack number at 57.6 Kb

SFC Reference

Appendix Objectives

Use this appendix to make sure your SFC meets your processor's requirements and to make sure your SFC runs the way you expect. This appendix discusses:

- SFC status information in the Processor Status file
- memory allocation
- dynamic constraints
- scanning sequences
- run times

SFC Status Information in the Processor Status File

Table B.A lists the words and bits in the processor status file (S) that contain SFC information.

Table B.A
SFC Status Words

Word:	Title:	Description:
S:1/15	First pass	Set: Processor began first program scan of the next active step in the SFC Reset: Processor completed scanning the currently active step
S:8	Current program scan time	The time for the processor to scan through all active steps one time If you are using multiple main control programs on a Enhanced PLC-5 processor, this time is the current total of one scan of all main control programs.
S:9	Maximum program scan time	The maximum time for the processor to scan through all active steps one time (word S:8) If you are using multiple main control programs on an Enhanced PLC-5 processor, this time is the maximum of all previous totals. This value is maintained until user resets it.
S:11/3	SFC fault	Set: Processor detected an SFC fault and stored a fault code in word 12 Reset: No SFC fault
S:11/5	Start up fault	Set: Processor detected a start-up protection fault (see word 26 bit 1) Reset: No fault, start up allowed

(Continued)

Word:	Title:	Description:	
S:12	Fault codes	74	Fault in SFC file
		75	SFC has more than 24 active steps
		77	Missing file or file of wrong type for step, action or transition
		78	SFC execution cannot continue after interruption
		79	Cannot run SFC because PLC-5 is incompatible
S:13	Faulted File Number	Contains the file number if an SFC fault occurred	
S:14	Faulted Rung Number	Contains the faulted rung number	
S:26/0 *	Restart/continue	Set:	Processor restarts SFC at the active steps where it left off due to power loss or processor mode change
		Reset:	Processor restarts SFC at first step
S:26/1 *	Start-up protection after power loss	Set:	Protection enabled; processor goes to fault routine at power up and processor sets word 11, bit 5
		Reset:	Protection disabled; processor powers up in run mode
S:28 *	Program watchdog setpoint	Maximum time (milliseconds) for scanning a single pass through all active steps	
		If you are using multiple main control programs on an Enhanced PLC-5 processor this time is the total of one scan of all main control programs.	
S:79 * (except for scan time) – S:127	MCP inhibit, file number and scan time	Information on the individual multiple main control programs. Enhanced PLC-5 processors only.	

* You enter values for these words/bits

Memory Allocation

The memory requirements for your SFC depend on the structures you use. Table B.B shows estimated word usage for SFC structures:

Table B.B
SFC Memory Usage

This Structure:	Uses this Amount of Memory:	
	Classic PLC-5 Processor	Enhanced PLC-5 Processor
start and end of program	2 words	19 words
each step /transition pair	8 words	16 + 6a words a = number of actions in step 6 words each action
each selection branch	5n + 5 words n = number of branches	11 + 6a + 7n a = number of actions in step n = number of paths
each simultaneous branch, diverging	n + 1 word n = number of branches	3n + 1 n = number of paths
each simultaneous branch, converging	n ² + 6n + 3 words n = number of branches	5 + 11n + 6a a = number of actions in all converging steps for that simultaneous branch n = number of paths
each label or GOTO statement	1 word	1 word
each chart compression	3 words	3 words

Figure B.1 shows a sample SFC and the estimated memory requirements for the SFC.

Figure B.1
Sample SFC and Memory Requirements

	<p>Classic PLC-5 Processors</p> <p>step/transition pair 8 words</p> <p>simultaneous diverge n = 2 n + 1 = 3 words</p> <p>selection branch n = 3 5n + 5 = 20</p> <p>3 step/transition pairs 3 x 8 = 24 words</p> <p>simultaneous converge n = 2 n² + 6n + 3 = 19 words</p> <p>step/transition 8 words</p> <hr/> <p>82 words (sub total) + 2 words (start and end of program)</p> <hr/> <p>84 words total for SFC</p>	<p>Enhanced PLC-5 Processors</p> <p>one action/step a=1 16 + 6a=22 words</p> <p>simultaneous diverge n = 2 3n + 1 = 7 words</p> <p>selection branch n = 3 a = 1 11 + 6a + 7n = 38 words</p> <p>3 step/transition pairs a=1 3 (16 + 6a) = 66 words</p> <p>simultaneous converge n = 2 a = 2 5 + 11n + 6a = 39 words</p> <p>one action/step a = 1 16 + 6a = 22 words</p> <hr/> <p>194 words (sub total) + 18 words (start and end of program) (8 actions * 6 words – assumes 1 unique action per step)</p> <hr/> <p>260 words total for SFC</p>
--	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Dynamic Constraints – Classic PLC-5 Processors Only

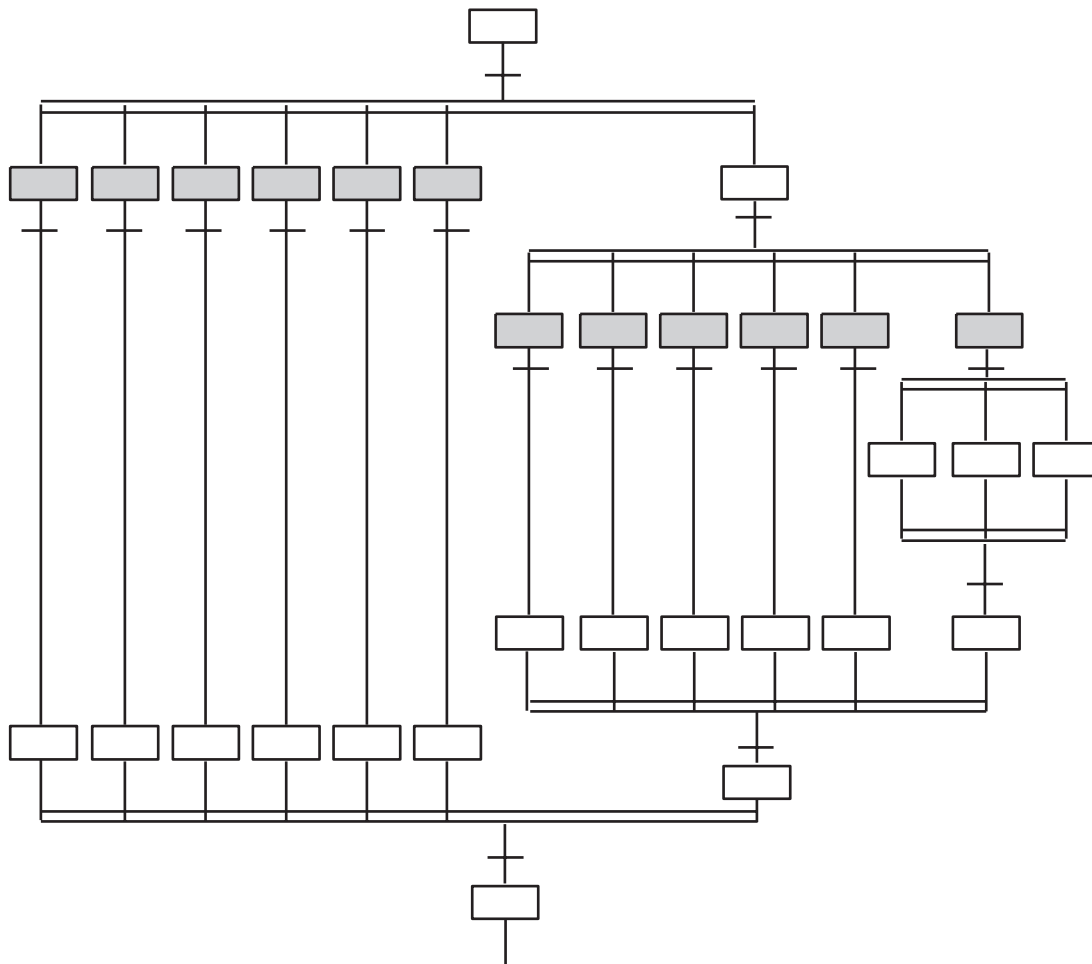
If you are using a Classic PLC-5 processor and your SFC has more than 12 parallel paths, you need to determine the number of parallel paths that could be active at one time. The dynamic limit is 24 parallel paths active at the same time for a Classic PLC-5 processor.

When a transition goes true, momentarily both the previously active step(s) (now waiting for postscan) and the newly active step(s) are on the execution queue together. You can have up to 23 parallel active steps as long as you can guarantee that no more than one transition goes true at one time.

Determine the number of active steps by counting the steps on each side of the transitions that control the widest area of the SFC. For example, 12 transitions that are true at the same time account for at least 24 simultaneous active steps. If any new simultaneous divergences follow one of these transitions, the maximum of 24 active paths is exceeded.

If the function chart in Figure B.2 is at the point where all 12 shaded steps are active and all of the transitions following those steps become true at the same time, the system attempts to have 26 active steps (12 for postscan, 14 for first scan) and the processor will fault.

Figure B.2
Dynamic Limit of Active Steps Could Be Exceeded
(Classic PLC-5 Processors)



Scanning Sequences

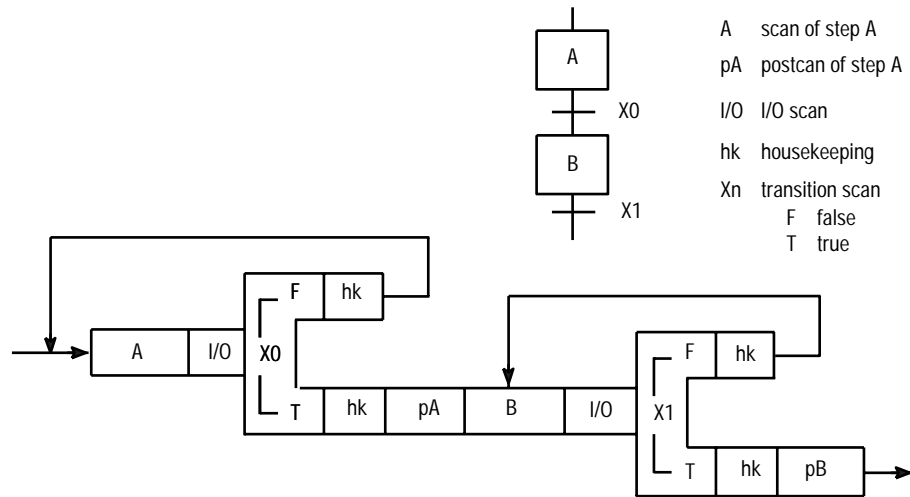
The processor scans the SFC from top to bottom, left to right. When the scan encounters active parallel steps, the processor runs the ladder logic in the left-most step first, then moves to the ladder logic in the next parallel step, until all active steps are run. The processor recognizes parallel steps by their position with respect to their common divergence, not necessarily by their position on the screen.

Step and Transition Scanning

In general, the processor scans an active step, then scans the I/O, and continues this cycle until the transition logic is true. Scanning the step includes evaluating all step action qualifiers and scanning all appropriate actions. When the transition is true, the processor scans the current step one more time (postscan). During postscan, the processor forces all rungs in the step false and resets rung logic. The processor does not update I/O between a postscan and the scan of the next active step. Figure B.3 shows the scan sequence for a step, transition and postscan. If you are using Enhanced PLC-5 processors, you can configure the scan and postscan operations. For more information, see your programming manual.

Important: Subcharts activated by a chart are scanned just prior to system housekeeping.

Figure B.3
Scan Sequence for a Step, Transition, and Postscan

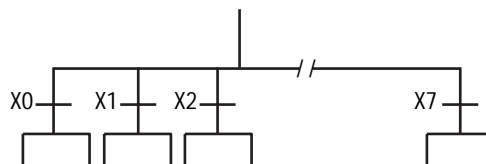


15556

Selected Branch Scanning

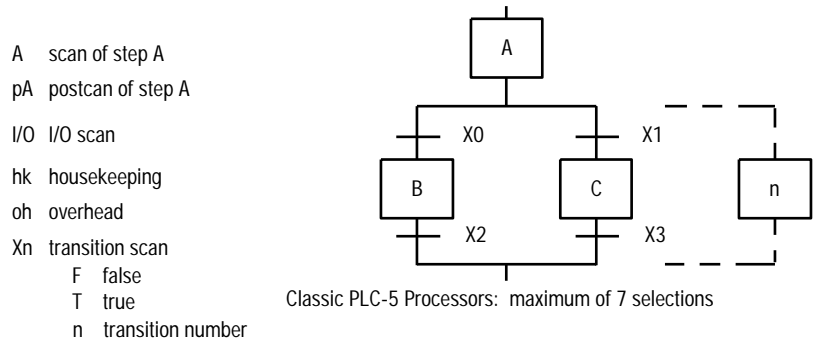
The processor selects one path of multiple parallel paths in a selected branch (Figure B.4). The processor tests transitions X0 through Xn, from left to right, until one of the transitions become true. The path with the first true transition is the active path.

Figure B.4
Selected Branch – Divergence



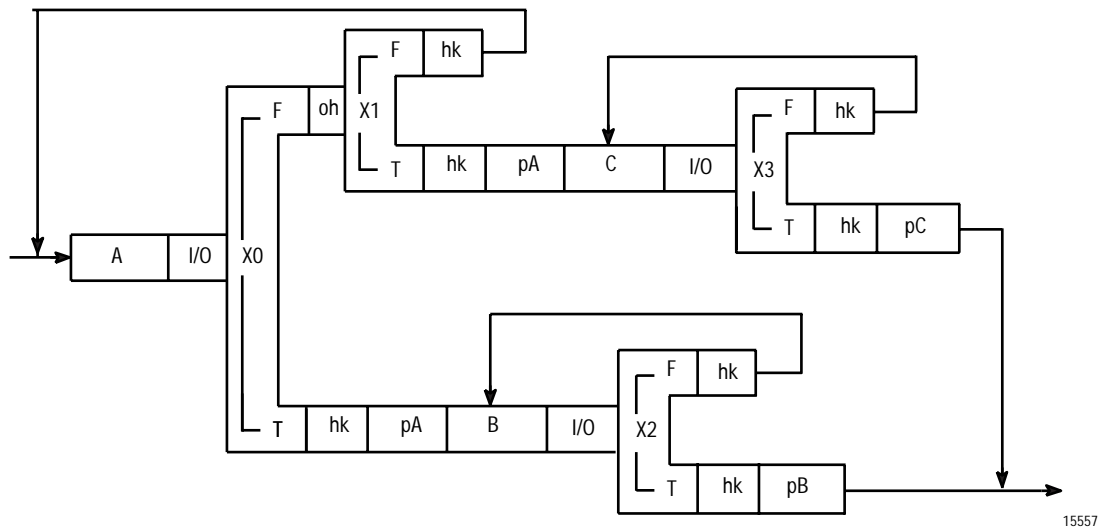
Because only one path is active, the scan sequence for the convergence is the same as for a step and transition. Figure B.5 shows the scan sequence for the divergence and convergence of a selected branch.

Figure B.5
Scan Sequence for a Selected Branch - Divergence and Convergence



Classic PLC-5 Processors: maximum of 7 selections

Enhanced PLC-5 Processors: maximum of 16 selections

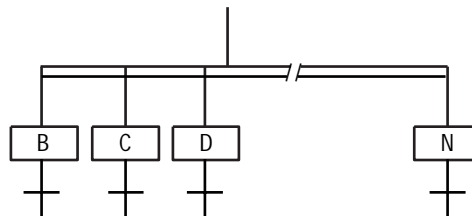


15557

Simultaneous Branch Scanning

The processor scans all parallel paths in a simultaneous branch (Figure B.6). On the first scan, the processor scans step B, then step C, until the processor scans all the steps on the divergence.

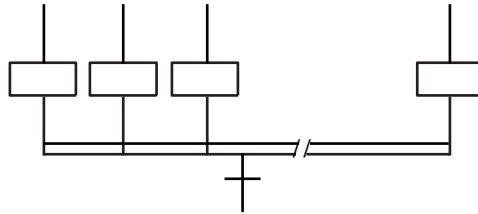
Figure B.6
Simultaneous Branch – Divergence



On subsequent scans, the processor scans in the order of step, I/O, and transition for each path, starting from the left.

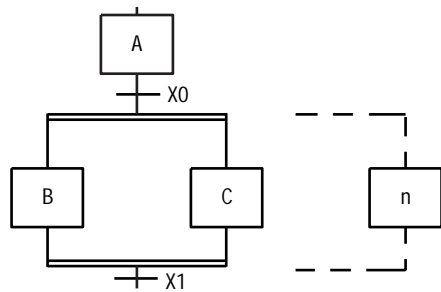
The vertical progression from step to step is independent of the active steps on the other parallel paths (Figure B.7).

Figure B.7
Simultaneous Branch - Convergence



The common transition cannot go true until the processor scans all the steps in the simultaneous branch at least once. Once the transition goes true, the processor does not scan the remaining paths in the branch; the processor postscans each step in the branch. Figure B.8 shows the scan sequence for the divergence and convergence of a selected branch.

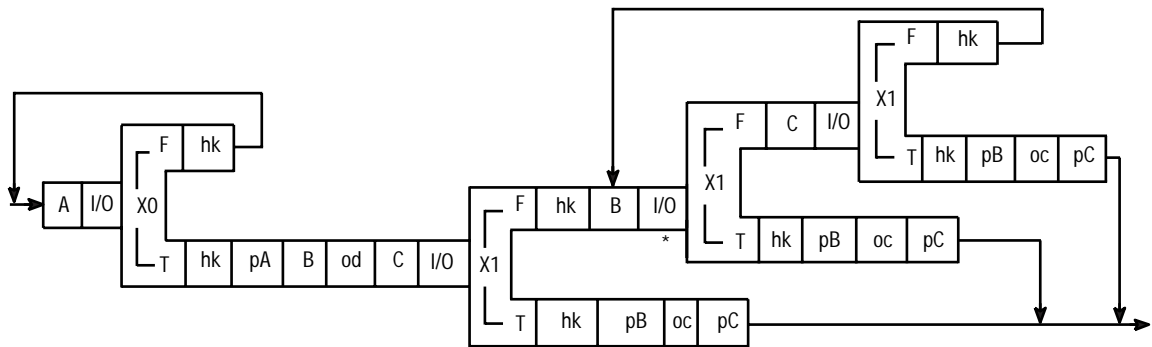
Figure B.8
Scan Sequence for a Simultaneous Branch – Divergence and Convergence



- A scan of step A
- pA postscan of step A
- I/O I/O scan
- hk housekeeping**
- Xn transition scan
- F false
- T true
- oc convergence overhead
- od divergence overhead

Classic Processors: maximum of 7 selections

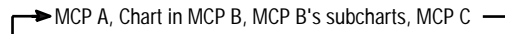
Enhanced PLC-5 Processors: maximum of 16 selections



15558

* In an Enhanced PLC-5 Processors, these states do not occur if scan configuration is set to ADVANCED mode.

** Any subcharts tied to this MCP execute now, followed by execution of subsequent MCPs. If this chart is MCP B and has active subchart actions while MCP A and C have ladder programs the sequence is:



SFC Example and Scan Sequence

Figure B.9 shows an example SFC. Figure B.10 shows the scan sequence for the example SFC. Use this example SFC and scan sequence as a guide. These figures may not apply to your system.

Figure B.9
Example SFC for Scan Sequence Example

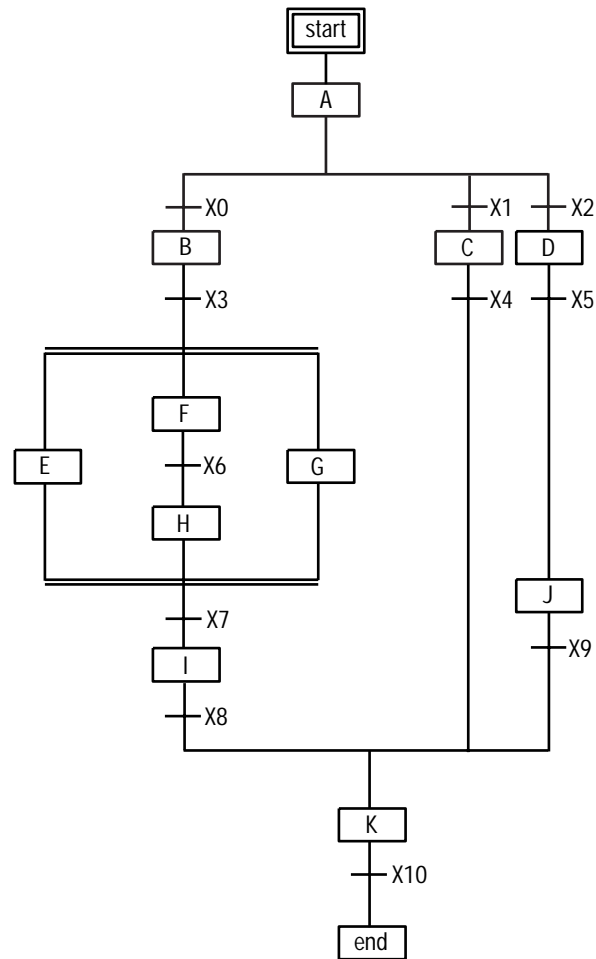
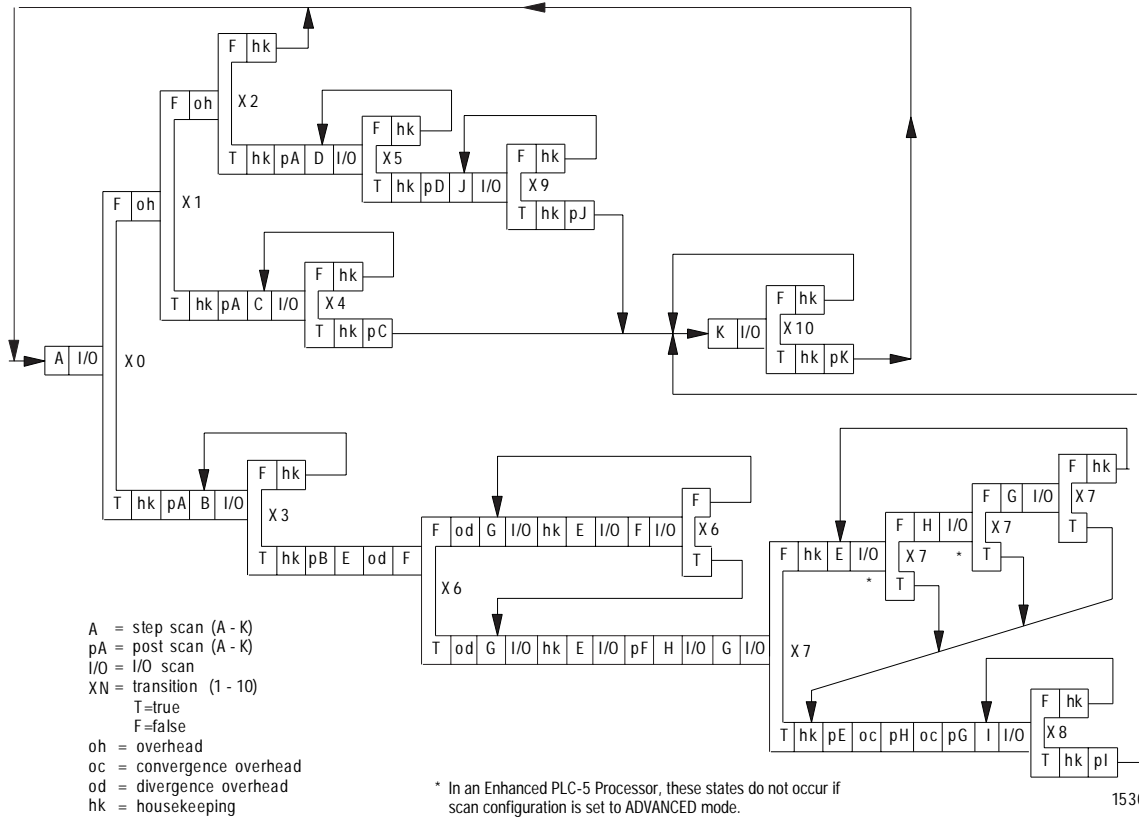


Figure B.10
Scan Sequence Example for the Example SFC



Run Times – Classic PLC-5 Processors

To determine the run time of your processor memory file on a Classic PLC-5 processor, you add the run time for ladder logic and the run time for the SFC. For information about run times for ladder logic, see appendix A. To determine the run time for an SFC, use either sequence diagrams or equations.

Using Sequence Diagrams to Determine Run Time

Table B.C lists the run times to add based on the sequence diagram for your SFC.

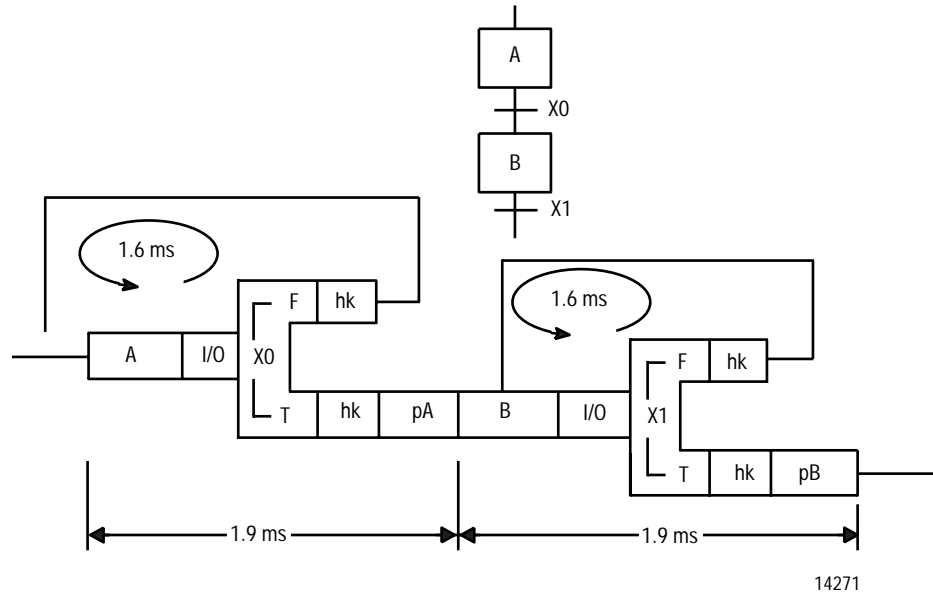
Table B.C
Run Times for Sequence Diagram Sections – Classic PLC-5 Processors

This Event:	Takes this Amount of Time (in milliseconds):
A	time to execute logic of step A + 0.1 ms
pA	time to scan logic of step A with rungs false + 0.1 ms
XN	transition N false (F): time to scan logic + 0.1 ms transition N true (T): time to scan logic + .25 ms
I/O (I/O scan)	0.6 ms
hk (housekeeping)	0.7 ms (increases with increasing DH+ traffic)
oh (overhead)	0.02 ms
od (divergence overhead)	0.3 ms
oc (convergence overhead)	0.2 ms

To determine the worst-case run time, assume that a transition goes true just after an I/O scan or just after a transition is scanned. This assumption requires an extra scan sequence before the transition goes true.

The scan time of a step and transition is proportional to the number of rungs for the step and transition. Figure B.11 shows the minimum scan time for a step that contains a single OTE and an END statement and a transition that contains a single XIC and an EOT statement.

Figure B.11
Minimum Scan Time for a Step and Transition Pair



Using Equations to Determine Run Time

The equations you use depend on whether the scan is steady state (simple step and transition) or divergent and convergent.

Steady-state Scan Time is when all transitions following active steps are false. Use this equation (Table B.D):

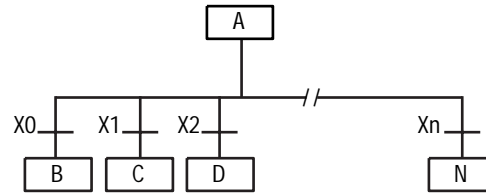
$$T_{\text{milliseconds}} = 0.8a + 0.7 + T_{\text{scan}}$$

Table B.D
Variables for Steady-State Scan Time

Where:	Is:
$T_{\text{milliseconds}}$	steady-state scan time in milliseconds
a	number of active steps
T_{scan}	total time to scan logic in all active steps and associated false transitions

Divergent Scan Time starts when the processor tests a transition and ends when the processor scans the next step's I/O. Divergent scan time includes transition scan time, postscan time of the previous step, scan time of the new step, overhead, and scan time of each parallel active step outside of the divergence.

For a selected-path divergence, the best case is when the transition goes true just before the I/O scan. Use this equation (Table B.E):

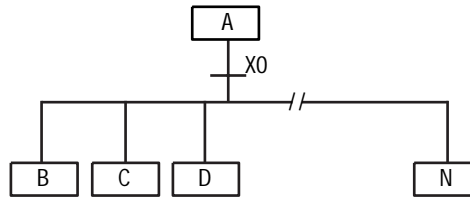


$$T_{\text{milliseconds}} = T_X + pA + T_S + 0.02(n-1) + 1.55 + 0.8a + T_0$$

Table B.E
Variables for Selected-Path Divergent Scan Time

Where:	Is:
$T_{\text{milliseconds}}$	transition scan time in milliseconds from step A to the first step in selected path N
T_X	sum of scan times of logic of transitions X0, X1, . . . , Xn in the divergence, up to and including the selected transition
pA	postscan time for the step (step A) preceding the divergence
T_S	scan time for logic in the new step (step N)
n	path number selected (1-7, from left to right)
a	number of active steps outside the divergence
T_0	sum of scan times of logic in all other active steps and transitions parallel to the divergence, but outside of the divergence

For a simultaneous divergence, the best case is when the transition goes true just before the I/O scan. Use this equation (Table B.F):



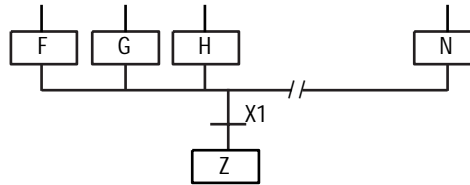
$$T_{\text{milliseconds}} = T_{X0} + pA + T_S + 0.3(n-1) + 1.97 + 0.8a + T_0$$

Table B.F
Variables for Simultaneous-Path Divergent Scan Time

Where:	Is:
$T_{\text{milliseconds}}$	transition time in milliseconds from when transition X0 goes true until the processor finishes scanning the last simultaneous step (step N) in the divergence
T_{X0}	scan time of logic in transition X0
pA	time to do a post-scan of step A
T_S	sum of scan times of logic in new steps (step B, step C, . . . , step N)
n	number of simultaneous active steps in the divergence
a	number of parallel active steps outside the divergence
T_0	sum of scan times of logic in all other active steps and transitions parallel to the divergence, but outside of the divergence

For the worst case, assume that a transition goes true just after the I/O scan or just after a transition is scanned. This assumption requires an extra scan sequence before the transition goes true.

Convergent Scan Time is when a simultaneous branch ends. The best case is when the transition goes true just before the I/O scan. Use this equation (Table B.G):



$$T_{\text{milliseconds}} = T_{X1} + T_p + T_z + 0.2(n-1) + 1.5 + 0.8a + T_0$$

Table B.G
Variables for Simultaneous-Path Convergent Scan Time

Where:	Is:
$T_{\text{milliseconds}}$	transition time in milliseconds from when transition X1 goes true until the processor finishes scanning step Z
T_{X1}	scan time of logic in transition X1
T_p	sum of postscan times of steps F, G, . . . , N
T_z	scan time of logic in step Z
n	number of simultaneous active steps in the convergence
a	number of parallel active steps outside of the convergence
T_0	sum of scan times of logic in all other active steps and transitions parallel to the convergence, but outside of the convergence

For the worst case, assume that a transition goes true just after the I/O scan or just after a transition is scanned. This assumption requires an extra scan sequence before the transition goes true.

Notes:

Valid Data Types for Instruction Operands

Appendix Objectives

This appendix lists all of the available instructions and their operands and the data types/values that are valid for each operand.

The following table explains each valid data type/value:

This Data Type/Value:	Accepts:
immediate (program constant)	any value between -32,768 and 32,767
integer	any integer data type: integer, timer, counter, status, bit, input, output, ASCII, BCD, control (e.g., N7:0, C4:0, etc.)
float	any floating point data type with 7-digit precision (valid range is $\pm 1.1754944e^{-38}$ to $\pm 3.4028237e^{+38}$).
block transfer	any block transfer data type (e.g., BT14:0)
ControlNet transfer	any CT data type (e.g., CT14:0)
message	any message data type (e.g., MG15:0)
PID	any PID data type (e.g., PD16:0) or integer data type (e.g., N7:0)
string	any string data type (e.g., ST12:0)
SFC status	any SFC status data type (e.g., SC17:0)

Instruction Operands and Valid Data Types

Table C.A shows the programming instructions you can use and the operands for those instructions. You can also use this table to format instructions in ASCII for importing. For more information on importing, see your programming manual.

Instructions marked with an asterisk (*) are only supported by Enhanced PLC-5 processors.

To enter the import syntax for any of the instructions listed in Table C.A:

- enclose all of the operands in parentheses
- separate each of the operands by commas

For example, the following is the import syntax for the FAL instruction:

```
FAL (R6:0, 10, 0, ALL, #N7:0, #N7:1+N7:2);
```

Table C.A
Programming Instructions and Operands

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
ABL *	ASCII Test Buffer for Line	channel	immediate, 0-4 integer	yes
		control	control	
ACB *	ASCII Number of Characters in Buffer	channel	immediate, integer	yes
		control	control	
ACI *	ASCII String to Integer	source	string	no
		destination	integer	
ACN *	ASCII String Concatenate	source A	string	no
		source B	string	
		destination	string	
ACS *	Arc Cosine	source	immediate, float (in radians), integer	no
		destination	float (in radians), integer	
ACT *	SFC action (only for ASCII import/export)	action number	immediate	N/A
		file number	0 - 999	
		destination	string	
ADD	ADD	source A	immediate, integer, float	no
		source B	immediate, integer, float	
		destination	integer, float	
AEX *	String Extract	source	string	no
		index	immediate, 0-82 integer	
		number	immediate, 0-82 integer	
		destination	string	
AFI	Always False	none		no
AHL *	ASCII Set/Reset Handshake Lines	channel	immediate, 0-4 integer	yes
		handshake AND mask	immediate, Hex integer	yes
		handshake OR mask	immediate, Hex integer	
		control	control	
AIC *	ASCII Integer to String	source	immediate, integer	no
		destination	string	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
AND	Logical AND	source A	integer	no
		source B	integer	
		destination	integer	
ARD *	ASCII Read Characters	channel	immediate, 0-4 integer	yes
		destination	string	
		control	control	
		string length	0 - 82	
ARL *	ASCII Read Line	channel	immediate, 0-4 integer	yes
		destination	string	
		control	control	
		string length	0 - 82	
ASC *	ASCII String Search	source	string	no
		index	immediate, 0-4 integer	
		search	string	
		result	integer	
ASN *	Arc Sine	source	immediate, float (in radians)	no
		destination	float (in radians)	
ASR *	ASCII String Compare	source A	string	no
		source B	string	
ATN *	Arc Tangent	source	immediate, float (in radians)	no
		destination	float (in radians)	
AVE *	Average File	file	integer, float	yes
		destination	integer, float	
		control	control	
		length	1 - 1000	
		position	0 - 999	
AWA *	ASCII Write with Append	channel	immediate, 0-4 integer	yes
		source	string	
		control	control	
		string length	0 - 82	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
AWT *	ASCII Write	channel	immediate, integer	yes
		source	string	
		control	control	yes
		length	0 - 82	
BRK	Break	none		no
BSL	Bit Shift Left	file	binary	yes
		control	control	
		bit address	bit	1 - 16000 (length in bits)
		length		
BSR	Bit Shift Right	file	binary	yes
		control	control	
		bit address	bit	1 - 16000 (length in bits)
		length		
BTD	Bit Distribute	source	immediate, integer	no
		source bit	immediate, (0 - 15) integer	
		destination	integer	
		destination bit	immediate (0 - 15)	
		length	immediate (1 - 16)	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
BTR ¹	Block Transfer Read	rack	00-277 octal	yes
		group	0-7	
		module	0-1	
		control block	block, integer	
		data file	integer	
		length	0, 1-64	
		continuous	YES, NO	
BTW ¹	Block Transfer Write	rack	00-277 octal	yes
		group	0-7	
		module	0-1	
		control block	block, integer	
		data file	integer	
		length	0, 1-64	
		continuous	YES, NO	
CIO	ControlNet I/O Transfer	control block	ControlNet transfer (1 - 64)	yes
CIR	Custom Input Routine (for use with CAR applications only)	program file number	immediate (2 - 999) for all processors	N/A
		input parameter list	immediate, integer, float	
		return parameter list	integer, float	
CLR	Clear	destination	integer, float	no
CMP	Compare	expression, relative expression, expression	expression using values or addresses with evaluators (for a list, see chapter 3 in this manual)	no
		EXE mnemonic (end of expression)	EXE	
		only for ASCII import		
COP	File Copy	source	array	no
		destination	array	
		length	immediate (1 - 1000)	

¹ In non-continuous mode, BTR and BTW ladder functions requires a false-to-true transition to execute. In continuous mode, once the rung goes true, BTR and BTW functions continue to execute regardless of rung condition. See page 15-8 for more information.

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
COR	Custom Output Routine (for use with CAR applications only)	program file number	immediate (2 - 999) for all processors	no
		input parameter list	immediate, integer, float	
		return parameter list	integer, float	
COS *	Cosine	source	immediate, float (in radians)	no
CPT	Compute	math expression	expression using values or immediate integer float addresses with evaluators (for a list, see chapter 4 in this manual)	no
		EXE mnemonic only for ASCII import	EXE	
		relative expression	addresses with evaluators (for a list, see chapter 4 in this manual)	
		destination	integer, float	
CTD	Count Down	counter	counter	yes
		PRE	-32,768 - +32,767	
		ACC	-32,768 - +32,767	
CTU	Count Up	counter	counter	yes
		PRE	-32,768 - +32,767	
		ACC	-32,768 - +32,767	
DDT	Diagnostic Detect	source array	binary	yes
		reference array	binary	
		result array	integer	
		compare control	control	
		length	1 - 16000 (length in bits)	
		position	0 - 15999	
		result control	control	
		length	1 - 1000	
position	0 - 999			
DEG *	Degree (convert radians to degrees)	source	immediate, float (in radians)	no
		destination	immediate, float (in degrees)	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
DFA	Diagnostic Fault Annunciator	control file	integer	
		number of I/O	immediate (8, 16, 32)	
		program file number	immediate (3-999)	
DIV	Divide	source A	immediate, integer, float	no
		source B	immediate, integer, float	
		destination	integer, float	
DTR	Data Transitional	source	immediate, integer	no
		mask	immediate, integer	
		reference	integer	
EOC	end of SFC compression (see SOC)	only for ASCII import/export		N/A
EOR	end of rung	only for ASCII import/export		N/A
EOT	end of transition	none		no
ESE	end SFC selection branch (see SEL)	only for ASCII import/export		N/A
EQU	Equal	source A	immediate, integer, float	no
		source B	immediate, integer, float	
EOP	end of SFC program	only for ASCII import/export		N/A
ERI	error on an input instruction	only in ASCII export files		N/A
ERO	error on an output instruction	only in ASCII export files		N/A
ESI	end SFC simultaneous branch (see SIM)	only for ASCII import/export		N/A
FAL	File Arithmetic/Logical	control	control	yes
		length	1 - 1000	
		position	0 - 999	
		mode	(INC, 1-1000, ALL)	
		destination	integer, float	
		math expression	indexed math instruction	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
FBC	File Bit Compare	source array	binary	yes
		reference array	binary	
		result array	integer	
		compare control	control	
		length	1 - 16000 (length in bits)	
		position	0 - 15999	
		result control	control	
		length	1 - 1000	
		position	0 - 999	
FFL	FIFO Load	source operand	immediate, indexed, integer	yes
		FIFO array	indexed, integer	
		FIFO control	control	
		length	1 - 1000	
		position	0 - 999	
FFU	FIFO Unload	FIFO array	indexed, integer	yes
		destination	indexed, integer	
		FIFO control	control	
		length	1 - 1000	
		position	0 - 999	
FLL	Fill File	source operand	immediate, integer, float	no
		destination array	array	no
		length	immediate (1 - 1000)	
FOR	For Loop	LBL number	integer	no
		index	integer	
		initial value	immediate, integer	
		terminal value	immediate, integer	
		step size	immediate, integer	
FRD	From BCD	source	immediate, integer	no
		destination	integer	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
FSC	File Search and Compare	control	control	yes
		length	1 - 1000	
		position	0 - 999	
		mode	immediate, integer (0, INC, 1-1000, ALL)	
		math expression	indexed math instruction	
GEQ	Greater Than or Equal To	source A	immediate, integer, float	no
		source B	immediate, integer, float	
GRT	Greater Than	source A	immediate, integer, float	no
		source B	immediate, integer, float	
IDI	Immediate Data Input	data file offset	immediate (0-999), integer	yes
		length	immediate (1-64), integer	
		destination	integer	
IDO	Immediate Data Output	data file offset	immediate (0-999), integer	yes
		length	immediate (1-64), integer	
		source	integer	
IIN	Immediate Input	I (input) word	immediate, integer PLC-5/10, 11, 12, 15, 20, 25, 30: 000-077 PLC-5/40, 40L: 000-157 PLC-5/60, 60L, 80, :000-237	no
IOT	Immediate Output	O (output) word	immediate, integer PLC-5/10, 11, 12, 15, 20, 25, 30: 000-077 PLC-5/40, 40L: 000-157 PLC-5/60, 60L, 80: 000-237	no
JMP	Jump	label number	immediate Classic PLC-5 processors: 0-31 Enhanced PLC-5 processors: 0-255	no
JSR	Jump to Subroutine	ladder program number	immediate (2 - 999)	no
		input parameter list	immediate, integer, float	
		return parameter list	integer float	
LAB	SFC label (import/export only)	file number	immediate Classic PLC-5 processors: 0-31 Enhanced PLC-5 processors: 0-255	N/A

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
LBL	LBL (ladder program label)	label number	immediate Classic PLC-5 processors: 0-31 Enhanced PLC-5 processors: 0-255	no
LEQ	Less Than or Equal To	source A	immediate, integer, float	no
		source B	immediate, integer, float	
LES	Less Than	source A	immediate, integer, float	no
		source B	immediate, integer, float	
LFL *	LIFO Load	source operand	immediate, indexed, integer	yes
		LIFO array	indexed, integer	
		LIFO control	control	
		length	1 - 1000	
		position	0 - 999	
LFU *	LIFO Unload	LIFO array	indexed, integer	yes
		destination	indexed, integer	
		LIFO control	control	
		length	1 - 1000	
		position	0 - 999	
LIM	Limit	low limit	immediate, integer, float	no
		test	immediate, integer, float	
		high limit	immediate, integer, float	
LN *	Natural Log	source	immediate, integer, float	no
		destination	float	
LOG *	Log to the Base 10	source	immediate, integer, float	no
		destination	float	no
MCR	Master Control Relay			no
MEQ	Mask Compare Equal To	source operand	immediate, integer	no
		source mask	immediate, integer	
		compare operand	immediate, integer	
MOV	Move	source	immediate, integer, float	no
		destination	integer, float	
MSG	Message	control block	message, integer	yes

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
MUL	Multiply	source A	immediate, integer, float	no
		source B	immediate, integer, float	
		destination	integer, float	
MVM	Masked Move	source operand	immediate, integer	no
		source mask	immediate, Hex integer	
		destination	integer	
NEG	Negate	source	immediate, integer, float	no
		destination	integer, float	
NEQ	Not Equal To	source A	immediate, integer, float	no
		source B	immediate, integer, float	
NOT	Logical NOT	source	immediate, integer	no
		destination	integer	
NSE	SFC next selection branch	only for ASCII import/export		N/A
NSI	SFC next simultaneous branch	only for ASCII import/export		N/A
NXT	Next (FOR Loop)	for label number	immediate Classic PLC-5 processors: 0-31 Enhanced PLC-5 processors: 0-255	no
OR	Logical OR	source A	immediate, bits integer	yes
		source B	immediate, bits integer	
		destination	integer	
OSF *	One Shot Falling	storage bit	bit	yes; requires a true-to-false transition to execute
		output bit	immediate (0 - 15)	
		output word	integer	
ONS	One Shot	source bit	bit	yes
OSR *	One Shot Rising	storage bit	bit	yes
		output bit	immediate (0 - 15)	
		output word	integer	
OTE	Output Energize	destination bit	bit	no
OTL	Output Latch	destination bit	bit	no
OTU	Output Unlatch	destination bit	bit	no

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
PID	PID	control block	PD	no
		control block	integer	yes
		pv value	integer	
		tieback value	immediate, integer	
		cv value	integer	
RAD *	Radian (convert degrees to radians)	source	immediate, float (in degrees)	no
		destination	float (in radians)	
REF	SFC reference (see LAB) (ASCII import/export only)	label number	immediate (0 - 255)	N/A
RES	Timer/Counter Reset		timer, counter, control	no
RET	Return	return parameter list	immediate, integer, float	no
RTO ²	Retentive Timer On	timer	timer	yes
		time base	immediate (0.01, 1.0)	
		PRE	0 - 32767	
		ACC	0 - 32767	
SBR	Subroutine	input parameter list	integer, float	no
SDS	Smart Directed Sequencer	control file	integer	no
		number of I/O	immediate (8, 16, 32)	
		program file number	immediate (3-999)	
SDZ	start of delete zone, unassembled edits	only in ASCII export files		N/A
SEL	SFC selection branch	only for ASCII import/export		N/A
SFR*	SFC reset	SFC file number	immediate (1 - 999)	no
		restart at step	immediate, integer	
SIM	SFC simultaneous branch	only for ASCII import		N/A

²This instruction requires periodic scans to be updated. See page 2-13 in this manual or your Structured Text User Manual for more information.

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
SIN *	Sine	source	immediate, float (in radians)	no
		destination	float (in radians)	
SIZ	start of insert zone, unassembled edits	only in ASCII export files		N/A
SOC	start of compression	only for ASCII import/export		N/A
SOP	SFC start of program	only for ASCII import/export		N/A
SOR	start of rung	only for ASCII import/export		N/A
SQI	Sequencer Input	file	integer, indexed	no
		mask	immediate, Hex indexed, integer	
		source	immediate, indexed, integer	
		control	control	
		length	1 - 1000	
		position	0 - 999	
SQL	Sequencer Load	file	integer, indexed	yes
		source	immediate, indexed, integer	
		control	control	
		length	1 - 1000	
		position	0 - 999	
SQO	Sequencer Output	file	integer, indexed	yes
		destination mask	immediate, indexed, integer	
		destination	indexed, integer	
		control	control	
		length	1 - 1000	
		position	0 - 999	
SQR	Square Root	source	immediate, integer, float	no
		destination	integer, float	
SRT *	Sort	sort file	integer, float	yes
		file control	control	
		length	1 - 1000	
		position	0 - 999	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
SRZ	start of replace zone, unassembled edits	only in ASCII export files		N/A
STP	SFC step (Classic PLC-5 Processors) (ASCII import/export only)	file number	2 - 999	N/A
STP *	SFC step (Enhanced PLC-5 Processors) (ASCII import/export only)	step timer file number	2 - 9999	N/A
		time base	immediate (0.01, 1.0)	
		qualifier	N, S, R, L, D, P1, P0, SL, SD, DS	
		action number (from ACT)	immediate	
		timer file number	timer	
		time base	immediate (0.01, 1.0)	
STD *	Standard Deviation	standard deviation file	integer, float	yes
		destination	integer, float	
		file control	control	
		length	1 - 1000	
		position	0 - 999	
SUB	Subtract	source A	immediate, integer, float	no
		source B	immediate, integer, float	
		destination	integer, float	
TAN *	Tangent	source	immediate, float (in radians)	no
		destination	float (in radians)	
TID *	Token ID (ASCII import/export only)	token ID number (must be unique per SFC file)	immediate	N/A
TND	Temporary End			no
TOD	To BCD	source	immediate, integer	no
		destination	integer	

Instruction	Description	Operand	Valid Value	Require False-to-True Transition
TOF ²	Timer Off Delay	timer	timer	yes: requires true-to-false transition to execute
TOF ²	Timer Off Delay	time base	immediate (0.01, 1.0)	yes: requires true-to-false transition to execute
		PRE	0 - 32767	
		ACC	0 - 32767	
TON ³	Timer On Delay	timer	timer	yes
		time base	immediate (0.01, 1.0)	
		PRE	0 - 32767	
		ACC	0 - 32767	
TRC	SFC transition (ASCII import/export only)	file number	2 - 999 for all processors	N/A
UID *	User Interrupt Disable			no
UIE *	User Interrupt Enable			no
XIC	Examine On	source bit	bit	no
XIO	Examine Off	source bit	bit	no
XOR	Exclusive Or	source A	immediate, bits integer	no
		source B	immediate, bits integer	
		destination	integer	
XPY *	X to the Power of Y	source A	immediate, integer	no
		source B	immediate, integer	
		destination	integer	

²This instruction requires periodic scans to be updated. See page 2-9 in this manual or page 2-10 in the Structured Text User Manual for more information.

³This instruction requires periodic scans to be updated. See page 1-14 in this manual or your Structured Text User Manual for more information.

Notes:

A

- ABL instruction 17-4
- ACB instruction 17-5
- ACI instruction 17-6
- ACN instruction 17-7
- ACS instruction 4-11
- ADD instruction 4-12
- Addition instruction
 - ADD 4-12
- AEX instruction 17-7
- AFI instruction 13-13
- AHL instruction 17-8
- AIC instruction 17-9
- Always False instruction 13-13
- AND instruction 5-2
- AND Operation instruction
 - AND 5-2
- Arc Cosine instruction
 - ACS 4-11
- Arc Sine instruction
 - ASN 4-13
- Arc Tangent instruction
 - ATN 4-14
- ARD instruction 17-10
- ARL instruction 17-12
- ASC instruction 17-14

ASCII

- ABL 17-4
- ACB 17-5
- ACI 17-6
- ACN 17-7
- AEX 17-7
- AHL 17-8
- AIC 17-9
- ARD 17-10
- ARL 17-12
- ASC 17-14
- ASR 17-15
- AWA 17-15
- AWT 17-17
- ASCII instructions, strings 17-3
- ASCII Integer to String instruction 17-9
- ASCII Read Characters instruction 17-10
- ASCII Read Line instruction 17-12
- ASCII Set Handshake Lines instruction 17-8
- ASCII String Compare instruction 17-15
- ASCII String Concatenate instruction 17-7
- ASCII String Extract instruction 17-7
- ASCII String Search instruction 17-14
- ASCII String to Integer instruction 17-6
- ASCII Write Append instruction 17-15
- ASCII Write instruction 17-17
- ASN instruction 4-13
- ASR instruction 17-15
- ATN instruction 4-14

Attention

- 32- to 16-bit conversion 4-10
 - AVE indexed address 4-16
 - change index value 13-6
 - control structure addressing 10-4
 - DTR online programming 10-8
 - entering input addresses 1-6
 - entering output addresses 1-7
 - FAL indexed address 9-2
 - FOR and NXT with output branches 13-5
 - FOR and NXT within branches 13-5
 - indexed addressing 8-2
 - jumped timers and counters 13-4
 - MCR zones
 - overlapping or nesting 13-2
 - timers and counters 13-2
 - modify status bits of BTR/BTW 15-6
 - MSG
 - status bits .ST and .EW 15-24
 - online programming with ONS 13-14
 - pairing stack instructions 11-6
 - PID
 - change engineering unit max 14-22
 - change engineering unit min 14-22
 - change inputs or units 14-19
 - changing scaling 14-6
 - resume last state 14-10
 - setting temperature limits 14-28
 - update time 14-21
 - placement of critical counters 2-15, 2-17
 - resetting TON and TOF 2-8, 2-20
 - SRT indexed address 4-27
 - status of BTR/BTW bits 15-7
 - STD indexed address 4-30
 - use of control address 12-3
 - using control addresses 8-2
 - using control addresses for instructions 11-2
- AVE instruction 4-15
- Average File instruction
AVE 4-15
- AWA instruction 17-15
- AWT instruction 17-17

B

- Bit Distribute instruction
BTD 7-2
- Bit Shift Left (BSL) instruction 11-2
- Bit Shift Right (BSR) instruction 11-2
- block transfer
 - BTR instruction 15-3
 - BTW instruction 15-3
 - direct communication mode 15-2
 - I/O scan mode 15-1
 - instructions 15-1
 - programming examples 15-15
 - timing 15-13, 15-14
- Block Transfer Read instruction
BTR 15-3
- Block Transfer Write instruction
BTW 15-3
- Break (BRK) instruction 13-5
- BRK instruction 13-5
- BSL instruction 11-2
- BSR instruction 11-2
- BTD instruction 7-2
- BTR instruction 15-3
- BTW instruction 15-3

C

- CAR utility 18-1
- CIO instruction 15-22
 - monitoring 15-24
 - status bits 15-24
 - using 15-23
- Classic PLC5 processors 1
- Clear instruction
CLR 4-17
- CLR instruction 4-17
- CMP
 - instruction 3-2

- compare
 - EQU 3-5
 - expression 3-2
 - GEQ 3-5, 3-6
 - instructions 3-2
 - length of expressions 3-3
 - LEQ 3-6
 - LES 3-7
 - NEQ 3-10
 - compute
 - ACS 4-11
 - ADD 4-12
 - ASN 4-13
 - ATN 4-14
 - AVE 4-15
 - CLR 4-17
 - COS 4-18
 - CPT 4-5
 - DEG 6-3
 - DIV 4-19
 - EOT 13-18
 - expression 4-5
 - FSC 9-14
 - functions 4-9
 - IOT 1-7
 - length of expressions 4-7
 - LN 4-20
 - LOG 4-21
 - MUL 4-22
 - NEG 4-23
 - ONS 13-14
 - order of operation 4-8
 - RAD 6-4
 - SIN 4-24
 - SQR 4-25
 - SRT 4-26
 - STD 4-28
 - SUB 4-31
 - TAN 4-32
 - XPY 4-33
 - Compute instruction
 - CPT 4-5
 - connecting to Ethernet PLC5 processors using hostnames 16-6
 - control file
 - example 8-2
 - ControlNet I/O transfer instruction 15-22
 - ControlNet PLC5 processors 1
 - convergent
 - scan time B-14
 - conversion
 - BCD 6-2
 - FRD 6-2
 - Convert from BCD instruction
 - FRD 6-2
 - Convert to BCD instruction
 - TOD 6-2
 - COP instruction 9-19
 - COS instruction 4-18
 - Cosine instruction
 - COS 4-18
 - Count Down instruction 2-17
 - Count Up instruction 2-15
 - counter
 - CTD 2-17
 - CTU 2-15
 - RES 2-20
 - counters
 - instructions 2-13
 - CPT instruction 4-5
 - CTD instruction 2-17
 - CTU instruction 2-15
 - custom application routine 18-1
- D**
- data files
 - manipulating 8-3
 - range of values C-1
 - data storage
 - I/O image files 1-2
 - Data Transitional instruction
 - DTR 10-8

- DDT instruction 10-2
- DEG instruction 6-3
- Degree instruction
 - DEG 6-3
- derivative smoothing 14-4
- DFA instruction 18-1
- diagnostic
 - DDT 10-2
 - DTR 10-8
 - FBC 10-2
 - parameters 10-4, 10-8
 - search mode 10-2
 - status 10-5
- Diagnostic Detect instruction
 - DDT 10-2
- Diagnostic Fault Annunciator Instruction 18-1
- diagnostic instructions 10-1
- direct communication
 - block transfer 15-2
- DIV instruction 4-19
- divergent
 - scan time B-14
- Divide instruction
 - DIV 4-19
- DTR instruction 10-8

E

- element manipulation
 - LIM 3-7
 - MEQ 3-9
 - MOV 7-3
 - MVM 7-4
- End of Transmission instruction
 - EOT 13-18
- engineering units
 - scaling 14-5
- Enhanced PLC5 processors 1
- EOT instruction 13-18
- EQU instruction 3-5

- Equal To instruction 3-5
- Ethernet PLC5 processors 1
- Examine Off instruction 1-3
- Examine On instruction 1-3
- expression
 - determining the length of 3-3, 4-7

F

- FAL logical instruction 9-12
- FBC instruction 10-2
- FFL instruction 11-5
- FFU instruction 11-5
- FIFO Load (FFL) instruction 11-5
- FIFO Unload (FFU) instruction 11-5
- file
 - search and compare operations 9-17
- File Arithmetic and Logic instruction
 - FAL 9-2
- File Bit Comparison instruction
 - FBC 10-2
- file concepts
 - control structure 8-2
 - manipulating data 8-3
 - modes of operation 8-5
- File Copy instruction
 - COP 9-19
- File Fill instruction
 - FLL 9-20
- file instructions
 - logical 9-12
- File Search and Compare instruction
 - FSC 9-14

files

arithmetic operations 9-7

copy operations 9-5

functions 9-14

instruction

COP 9-19

FLL 9-20

logical operations 9-12

operation modes 8-5

FLL instruction 9-20

floating point

valid value range C-1

For (FOR) instruction 13-5

FOR instruction 13-5

FRD instruction 6-2

FSC instruction 9-14

G

gain constants 14-3

GEQ instruction 3-5, 3-6

Greater Than or Equal To instruction 3-5, 3-6

I

I/O image files 1-2

I/O scan mode

block transfer 15-1

IDI instruction 1-8

using 1-9

IDO instruction 1-8

using 1-9

IIN instruction 1-6

Immediate Data Input

instruction 1-8

Immediate Data Output

instruction 1-8

Immediate Input instruction 1-6

Immediate Output instruction

IOT 1-7

incremental mode 8-7

instruction

ControlNet I/O transfer 15-22

immediate data input 1-8

immediate data output 1-8

instructions

ASCII 17-1

block transfer 15-1

CIO

monitoring 15-24

compare 3-2

diagnostic 10-1

memory requirements A-1

message 16-1

operands C-1

program flow 13-1

relay-type 1-1, 2-1

sequencer 12-1

shift register 11-1

timer 2-1

timing A-1

INVALID OPERAND

error message 4-4

IOT instruction 1-7

J

JMP instruction 13-3

JSR instruction 13-8

Jump instruction 13-3

Jump to Subroutine instruction 13-8

L

Label (LBL) instruction 13-5

Label instruction 13-3

LBL instruction 13-3, 13-5

LEQ instruction 3-6

LES instruction 3-7

Less Than instruction 3-7

Less Than or Equal To instruction 3-6

LFL instruction 11-5

LFU instruction 11-5

- LIFO Load (LFL) instruction 11-5
- LIFO Unload (LFU) instruction 11-5
- LIM instruction 3-7
- Limit Test instruction 3-7
- LN instruction 4-20
- LOG
 - instruction 4-21
- Log to the base 10 instruction
 - LOG 4-21
- logical
 - AND 5-2
 - NOT 5-3
 - OR 5-4
 - XOR 5-5

- M**
- manipulating
 - file data 8-3
- Masked Comparison for Equal instruction 3-9
- Masked Move instruction 7-4
- Master Control Reset instruction 13-2
- MCR instruction 13-2
- memory
 - instruction requirements A-1
 - SFC requirements B-3
- MEQ instruction 3-9
- message
 - instruction 16-1
- modes
 - file operation 8-5
- monitoring
 - CIO instructions 15-24
- MOV instruction 7-3
- Move instruction
 - MOV 7-3
- MSG
 - instruction entry 16-10
- MSG instruction 16-1
 - using 16-10

- MUL instruction 4-22
- Multiply instruction
 - MUL 4-22
- MVM instruction 7-4

- N**
- Natural Log instruction
 - LN 4-20
- NEG instruction 4-23
- Negate instruction
 - NEG 4-23
- NEQ instruction 3-10
- Next (NXT) instruction 13-5
- Not Equal To instruction 3-10
- NOT instruction 5-3
- NOT Operation instruction
 - NOT 5-3
- Number of Char in Buffer instruction 17-5
- NXT instruction 13-5

- O**
- One Shot Falling (OSF) instruction 13-16
- One Shot instruction
 - ONS 13-14
- One Shot Rising (OSR) instruction 13-15
- ONS instruction 13-14
- operands
 - instructions C-1
- OR instruction 5-4
- OR Operation instruction
 - OR 5-4
- order of operation 4-8
- OSF instruction 13-16
- OSR instruction 13-15
- OTE instruction 1-4
- OTL instruction 1-4
- OTU instruction 1-5

Output Energize instruction 1-4

Output Latch instruction 1-4

Output Unlatch instruction 1-5

P

PID

 biasing 14-9

 equations 14-2

 examples 14-29

 instruction 14-10

 integer examples 14-29

 PD examples 14-33

 selecting derivative term 14-7

 setting output alarms 14-7

 using manual mode 14-8

 using output limiting 14-7

PID instruction 14-1

PLC2 compatibility file 16-15

process control

 biasing 14-9

 derivative smoothing 14-4

 equations 14-2

 gain constants 14-3

 integer PID examples 14-29

 PD PID examples 14-33

 PID 14-10

 PID examples 14-29

 PID instruction 14-1

 selecting derivative term 14-7

 setting output alarms 14-7

 using manual mode 14-8

 using output limiting 14-7

program constant

 valid value range C-1

program constants 3-2, 4-5

program flow

 AFI 13-13

 JMP and LBL 13-3

 JSR, SBR, and RET 13-8

 MCR 13-2

 UID 13-19

 UIE 13-20

program flow instruction

 FOR, BRK, LBL, and RET 13-5

 OSF 13-16

 OSR 13-15

 SFR 13-17

program flow instructions 13-1

Programming

 SDS instruction 18-2

programming

 instructions

 operands C-1

Proportional, Integral, and Derivative instruction

 14-10

R

RAD instruction 6-4

Radian instruction

 RAD 6-4

relay-type

 IIN 1-6

 OTE 1-4

 OTL 1-4

 OTU 1-5

 XIC 1-3

 XIO 1-3

RES instruction 2-20

Reset instruction 2-20

RET instruction 13-8

Retentive Timer On instruction 2-10

Return instruction 13-8

RTO instruction 2-10

run times

 determining B-12

S

- SBR instruction 13-8
- scaling
 - to engineering units 14-5
- scan sequence
 - SFC B-7
- scan time
 - convergent B-14
 - divergent B-14
 - steady-state B-14
- scanner mode
 - configuring 15-13, 15-14
- SDS instruction 18-1
- selection branch
 - scanning sequence B-8
- sequencer
 - applying 12-1
 - instructions 12-1
 - SQI 12-2
 - SQL 12-2
 - SQO 12-2
- Sequencer Input instruction 12-2
- Sequencer Load instruction 12-2
- Sequencer Output instruction 12-2
- Sequential Function Chart Reset instruction 13-17
- SFC
 - constraints B-5
 - example
 - scanning sequence B-11
 - memory requirements B-3
 - scanning sequence
 - example B-11
 - selection branch B-8
 - simultaneous branch B-9
 - scanning sequences
 - step/transition B-7
 - status information B-1
- SFR instruction 13-17
- shift register instruction
 - applying 11-1
 - BSL and BSR 11-2
 - FFL and FFU 11-5
 - LFL and LFU 11-5
- simultaneous branch
 - scanning sequence B-9
- SIN instruction 4-24
- Sine instruction
 - SIN 4-24
- Smart Directed Sequencer (SDS) Instruction
 - overview 18-2
 - programming 18-2
- Smart Directed Sequencer Instruction 18-1
- Sort File instruction
 - SRT 4-26
- SQI instruction 12-2
- SQL instruction 12-2
- SQO instruction 12-2
- SQR instruction 4-25
- Square Root instruction
 - SQR 4-25
- SRT instruction 4-26
- Standard Deviation instruction
 - STD 4-28
- status bits
 - CIO instruction 15-24
- status information
 - SFC B-1
- STD instruction 4-28
- steady-state
 - scan time B-14
- step
 - scanning sequence B-7
- SUB instruction 4-31
- Subroutine Header instruction 13-8
- Subtract instruction
 - SUB 4-31

T

- TAN instruction 4-32
- Tangent instruction
 - TAN 4-32
- Temporary End
 - instruction 13-13
- Temporary End instruction 13-20
- Test Buffer For Line instruction 17-4
- timer
 - accuracy 2-3
 - instruction parameters 2-2, 2-13
 - RES 2-20
 - RTO 2-10
 - TOF 2-7
 - TON instruction 2-4
- Timer Off Delay instruction 2-7
- Timer On Delay instruction 2-4
- timers 2-1
- timing
 - block transfer 15-13, 15-14
 - instructions A-1
- tip
 - connecting to Ethernet PLC5 processors
 - using hostnames 16-6
- TND
 - instruction 13-13
- TND instruction 13-19, 13-20
- TOD instruction 6-2
- TOF instruction 2-7
- TON instruction 2-4
- transition
 - scanning sequence B-7

U

- units, engineering
 - scaling 14-5
- User Interrupt Disable
 - UID 13-19
- User Interrupt Enable
 - UIE 13-20
- using
 - CIO instruction 15-23
 - IDI instruction 1-9
 - IDO instruction 1-9
 - MSG instruction 16-10

X

- X to the Power of Y instruction
 - XPY 4-33
- XIC instruction 1-3
- XIO instruction 1-3
- XOR instruction 5-5
- XOR Operation instruction
 - XOR 5-5
- XPY instruction 4-33

Notes:



Allen-Bradley Publication Problem Report

If you find a problem with our documentation, please complete and return this form

Pub. Name PLC-5 Programmable Controllers Instruction Set Reference

Cat. No. 1785 series Pub. No. 1785-6.1 Pub. Date November 1998 Part No. 955133-83

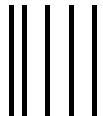
Check Problem(s) Type:	Describe Problem(s):	Internal Use Only
<input type="checkbox"/> Technical Accuracy	<input type="checkbox"/> text <input type="checkbox"/> illustration	
<input type="checkbox"/> Completeness What information is missing?	<input type="checkbox"/> procedure/step <input type="checkbox"/> illustration <input type="checkbox"/> definition <input type="checkbox"/> example <input type="checkbox"/> guideline <input type="checkbox"/> feature <input type="checkbox"/> explanation <input type="checkbox"/> other	<input type="checkbox"/> info in manual (accessibility) <input type="checkbox"/> info not in manual
<input type="checkbox"/> Clarity What is unclear?		
<input type="checkbox"/> Sequence What is not in the right order?		
<input type="checkbox"/> Other Comments Use back for more comments.		
Your Name _____	Location/Phone _____	

Return to: Marketing Communications, Allen-Bradley Co., 1 Allen-Bradley Drive, Mayfield Hts., OH 44124-6118
 Phone: (440)646-3166
 FAX: (440)646-4320

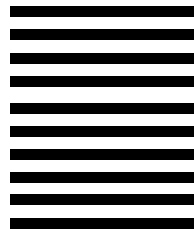
Other Comments

Lined area for additional comments or notes.

PLEASE FOLD HERE



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



PLEASE REMOVE

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE



Allen-Bradley

1 ALLEN BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705



Customer Support

If you need additional assistance in using your software, Allen-Bradley offers telephone and on-site product support at Customer Support Centers worldwide.

For technical assistance on the telephone, first contact your local sales office, distributor, or system integrator. If additional assistance is needed, then contact your local Customer Support Center or contact System Support Services.

In the United States and Canada

If you have a SupportPlus agreement or your software is under warranty, you can contact System Support Services at: 1-800-289-2279. Have your support contract or software registration number available.

For assistance that requires on-site support, contact your local sales office, distributor, or system integrator. During non-office hours, contact Allen-Bradley 24-hour Hot Line at 1-800-422-4913.

Outside of the United States

Contact your local Customer Support Center at:

Region or Area	Customer Support Center Telephone Number
Canada (Cambridge, Ontario)	519-623-1810
Latin America (Mexico)	52-5-259-0040
United Kingdom (Milton Keynes)	44-908 838800
France (Paris)	(33-1) 3067-7200
Germany (Gruiten)	(49) 2104 6900
Italy (Milan)	(39-2) 939-721
Asia Pacific (Hong Kong)	(852) 887-4788
Spain (Barcelona)	(34-3) 331-7004

For assistance that requires on-site support, contact your local office, distributor, or system integrator. During non-office hours, contact your local Customer Support Center.



Rockwell Automation helps its customers receive a superior return on their investment by bringing together leading brands in industrial automation, creating a broad spectrum of easy-to-integrate products. These are supported by local technical resources available worldwide, a global network of system solutions providers, and the advanced technology resources of Rockwell.

Worldwide representation.



Argentina • Australia • Austria • Bahrain • Belgium • Bolivia • Brazil • Bulgaria • Canada • Chile • China, People's Republic of • Colombia • Costa Rica • Croatia • Cyprus • Czech Republic • Denmark • Dominican Republic • Ecuador • Egypt • El Salvador • Finland • France • Germany • Ghana • Greece • Guatemala • Honduras • Hong Kong • Hungary • Iceland • India • Indonesia • Iran • Ireland • Israel • Italy • Jamaica • Japan • Jordan • Korea • Kuwait • Lebanon • Macau • Malaysia • Malta • Mexico • Morocco • The Netherlands • New Zealand • Nigeria • Norway • Oman • Pakistan • Panama • Peru • Philippines • Poland • Portugal • Puerto Rico • Qatar • Romania • Russia • Saudi Arabia • Singapore • Slovakia • Slovenia • South Africa, Republic of • Spain • Sweden • Switzerland • Taiwan • Thailand • Trinidad • Tunisia • Turkey • United Arab Emirates • United Kingdom • United States • Uruguay • Venezuela

Rockwell Automation Headquarters, 1201 South Second Street, Milwaukee, WI 53204 USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444

Rockwell Automation European Headquarters SA/NV, avenue Hermann Debrouckelaan, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 05 40

Rockwell Automation Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846