GE Fanuc Intelligent Platforms
Programmable Control Products

PACSystems™

CPU Reference Manual, GFK-2222L

November 2008



Warnings, Cautions, and Notes as Used in this Publication

Warning

Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in this equipment or may be associated with its use.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where equipment might be damaged if care is not taken.

Note: Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all hardware and software systems. GE Fanuc assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Fanuc makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. No warranties of merchantability or fitness for purpose shall apply.

The following are trademarks of GE Fanuc Intelligent Platforms:

Alarm Master	Genius	ProLoop	Series Six
CIMPLICITY	Helpmate	PROMACRO	Series Three
CIMPLICITY 90–ADS	Logicmaster	PowerMotion	VersaMax
CIMSTAR	Modelmaster	PowerTRAC	VersaPoint
Field Control	Motion Mate	Series 90	VersaPro
GEnet	PACSystems	Series Five	VuMaster
	Proficy	Series One	Workmaster

©Copyright 2003—2008 GE Fanuc Intelligent Platforms North America, Inc.
All Rights Reserved

Contents

Introduction 1-1

New Features 1-2
PACSystems Control System Overview 1-3
Programming and Configuration 1-3
Process Systems 1-3
PACSystems CPU Models 1-4
RX3i Overview 1-5
RX7i Overview 1-5
Migrating Series 90 Applications to PACSystems 1-7
PACSystems Documentation 1-8
Technical Support 1-9
General Contact Information 1-9
Technical Support 1-9
Americas 1-9
Europe, the Middle East, and Africa 1-9
Asia Pacific 1-9

CPU Features and Specifications 2-1

Common CPU Features 2-1
Firmware Storage in Flash Memory 2-1
Operation, Protection, and Module Status 2-1
Ethernet Global Data 2-1
RX7i Features and Specifications 2-2
CPE010, CPE020 and CRE020 Models 2-2
CPE030/CRE030 and CPE040/CRE040 Models 2-5
Embedded Ethernet Interface 2-8
RX3i Features and Specifications 2-12
IC695CPU310 2-12
IC695CPU320 2-14

CPU Configuration 3-1

Configuring the CPU 3-1
Configuration Parameters 3-2
Settings Parameters 3-2
Modbus TCP Address Map 3-3
Scan Parameters 3-4
Memory Parameters 3-6
Fault Parameters 3-8
Redundancy Parameters (Redundancy CPUs Only) 3-10
Transfer List 3-10
Port 1 and Port 2 Parameters 3-11
Scan Sets Parameters 3-15
Power Consumption Parameters 3-15

Setting a Temporary IP Address ... 3-16
Storing (Downloading) Hardware Configuration ... 3-17

Embedded Ethernet Interface Configuration ... 4-1

Configuring the Embedded Ethernet Interface ... 4-2
Configuration Parameters ... 4-3
Ethernet Settings ... 4-3
RS-232 Port (Station Manager) Parameters ... 4-5
Terminals Tab ... 4-5
Verifying Proper Power-Up of the Ethernet Interface ... 4-7
Pinging TCP/IP Ethernet Interfaces on the Network ... 4-7
Pinging the Interface from a UNIX Host or a PC Running TCP/IP Software ... 4-7
Determining if an IP Address is Already Being Used ... 4-7

CPU Operation ... 5-1

CPU Sweep ... 5-2
Parts of the CPU Sweep ... 5-3
CPU Sweep Modes ... 5-6
Program Scheduling Modes ... 5-9
Window Modes ... 5-9
Data Coherency in Communications Windows ... 5-9
Run/Stop Operations ... 5-10
CPU Stop Modes ... 5-10
Stop-to-Run Mode Transition ... 5-11
Run/Stop Mode Switch Operation ... 5-12
Flash Memory Operation ... 5-13
Logic/Configuration Source and CPU Operating Mode at Power-up ... 5-14
Clocks and Timers ... 5-16
Elapsed Time Clock ... 5-16
Time-of-Day Clock ... 5-16
Watchdog Timer ... 5-18
System Security ... 5-19
Passwords and Privilege Levels ... 5-19
OEM Protection ... 5-20
PACSystems I/O System ... 5-21
I/O Configuration ... 5-21
Genius I/O ... 5-23
Genius Global Data Communications ... 5-24
I/O System Diagnostic Data Collection ... 5-24
Power-Up and Power-Down Sequences ... 5-26
Power-Up Sequence ... 5-26
Power-Down Sequence ... 5-28
Retention of Data Memory Across Power Failure ... 5-28

Program Organization 6-1

Structure of a PACSystems Application Program 6-1
Blocks 6-1
Functions and Function Blocks 6-1
How Blocks Are Called 6-2
Nested Calls 6-2
Types of Blocks 6-3
Local Data 6-12
Parameter Passing Mechanisms 6-13
Languages 6-14
Controlling Program Execution 6-16
Interrupt-Driven Blocks 6-17
Interrupt Handling 6-17
Timed Interrupts 6-18
I/O Interrupts 6-19
Module Interrupts 6-19
Interrupt Block Scheduling 6-19

Program Data 7-1

Variables 7-2
Mapped Variables 7-2
Symbolic Variables 7-2
I/O Variables 7-3
Reference Memory 7-6
Word (Register) References 7-6
Bit (Discrete) References 7-8
User Reference Size and Default 7-9
%G User References and CPU Memory Locations 7-9
Genius Global Data 7-10
Transitions and Overrides 7-11
Retentiveness of Logic and Data 7-12
Data Scope 7-13
System Status References 7-14
%S References 7-14
%SA, %SB, and %SC References 7-15
Fault References 7-17
How Program Functions Handle Numerical Data 7-19
Data Types 7-19
Floating Point Numbers 7-20
Word-for-Word Changes 7-23
Operands for Instructions 7-24

Ladder Diagram Programming 8-1

Advanced Math Functions 8-2
Exponential/Logarithmic Functions 8-3
Square Root 8-4
Trig Functions 8-5
Inverse Trig – ASIN, ACOS, and ATAN 8-6
Bit Operation Functions 8-7
Data Lengths for the Bit Operation Functions 8-8
Bit Position 8-9
Bit Sequencer 8-10
Bit Set, Clear 8-13
Bit Test 8-14
Logical AND, Logical OR, and Logical XOR 8-15
Logical NOT 8-18
Masked Compare 8-19
Rotate Bits 8-22
Shift Bits 8-23
Coils 8-25
Coil Checking 8-25
Graphical Representation of Coils 8-25
Set, Reset Coil 8-26
Transition Coils 8-28
Contacts 8-31
Continuation Contact 8-32
Fault Contact 8-32
High and Low Alarm Contacts 8-33
No Fault Contact 8-33
Normally Closed and Normally Open Contacts 8-34
Transition Contacts 8-35
Control Functions 8-39
Do I/O 8-40
Edge Detectors 8-43
Drum 8-45
For Loop 8-48
Mask I/O Interrupt 8-51
Read Switch Position 8-52
Scan Set IO 8-53
Suspend I/O 8-55
Suspend or Resume I/O Interrupt 8-57
Conversion Functions 8-58
Convert Angles 8-59
Convert UINT or INT to BCD4 8-59
Convert DINT to BCD8 8-60
Convert BCD4, UINT, DINT, or REAL to INT 8-61

Convert BCD4, INT, DINT, or REAL to UINT ... 8-63
Convert BCD8, UINT, INT, REAL or LREAL to DINT ... 8-65
Convert BCD4, BCD8, UINT, INT, DINT, and LREAL to REAL ... 8-67
Convert REAL to LREAL ... 8-69
Convert DINT to LREAL ... 8-69
Truncate ... 8-70
Counters ... 8-71
Down Counter ... 8-72
Up Counter ... 8-73
Data Move Functions ... 8-75
Block Clear ... 8-76
Block Move ... 8-77
BUS_ Functions ... 8-78
Communication Request ... 8-84
Data Initialization ... 8-89
Data Initialize ASCII ... 8-90
Data Initialize Communications Request ... 8-91
Data Initialize DLAN ... 8-91
Move Data ... 8-92
Shift Register ... 8-94
Swap ... 8-96
Data Table Functions ... 8-97
Array Move ... 8-99
Math Functions ... 8-115
Absolute Value ... 8-116
Add ... 8-117
Divide ... 8-119
Modulus ... 8-120
Multiply ... 8-121
Scale ... 8-123
Subtract ... 8-124
Program Flow Functions ... 8-125
Argument Present ... 8-125
Call ... 8-127
Comment ... 8-130
Jump ... 8-131
Master Control Relay/End Master Control Relay ... 8-132
Wires ... 8-134
Relational Functions ... 8-135
Compare ... 8-136
Equal, Not Equal, Greater or Equal, Greater Than, Less or Equal, Less Than ... 8-137
Range ... 8-138
Timers ... 8-139
Timed Contacts ... 8-139
Timer Function Blocks ... 8-140

Built-In Timer Function Blocks 8-140
Standard Timer Function Blocks 8-151

Function Block Diagram 9-1

Advanced Math Functions 9-2
EXPT Function 9-3
Bit Operation Functions 9-4
Logical AND, Logical OR, and Logical XOR 9-6
Logical NOT 9-8
Comments 9-9
Text Block 9-9
Comparison Functions 9-10
Equal, Not Equal, Greater or Equal, Greater Than, Less or Equal, Less Than 9-12
Control Functions 9-13
Counters 9-15
Data Move Functions 9-16
Fan Out 9-18
Move Data 9-18
Math Functions 9-21
Add 9-23
Divide 9-24
Modulus 9-25
Multiply 9-26
Negate 9-27
Subtract 9-28
Program Flow Functions 9-29
Timers 9-30
Built-in Timer Function Blocks 9-30
Standard Timer Function Blocks 9-31
Type Conversion Functions 9-32
Convert WORD to INT 9-34
Convert WORD to UINT 9-35
Convert DWORD to DINT 9-35
Convert INT or UINT to WORD 9-36
Convert DINT to DWORD 9-36

Service Request Function 10-1

Operation of SVC_REQ Function 10-2
Ladder Diagram 10-2
Function Block Diagram 10-3
SVC_REQ 1: Change/Read Constant Sweep Timer 10-4
SVC_REQ 2: Read Window Modes and Times Values 10-6
SVC_REQ 3: Change Controller Communications Window Mode 10-7
SVC_REQ 4: Change Backplane Communications Window Mode and Timer Value 10-8

SVC_REQ 5: Change Background Task Window Mode and Timer Value 10-9
SVC_REQ 6: Change/Read Number of Words to Checksum 10-10
SVC_REQ 7: Read or Change the Time-of-Day Clock 10-12
Parameter Block Formats 10-12
SVC_REQ 8: Reset Watchdog Timer 10-19
SVC_REQ 9: Read Sweep Time from Beginning of Sweep 10-19
SVC_REQ 10: Read Target Name 10-20
SVC_REQ 11: Read Controller ID 10-21
SVC_REQ 12: Read Controller Run State 10-22
SVC_REQ 13: Shut Down (Stop) CPU 10-23
SVC_REQ 14: Clear PLC or I/O Fault Table 10-24
SVC_REQ 15: Read Last-Logged Fault Table Entry 10-25
SVC_REQ 16: Read Elapsed Time Clock 10-27
SVC_REQ 17: Mask/Unmask I/O Interrupt 10-29
Masking/Unmasking Module Interrupts 10-29
SVC_REQ 18: Read I/O Forced Status 10-31
SVC_REQ 19: Set Run Enable/Disable 10-32
SVC_REQ 20: Read Fault Tables 10-33
Non-Extended Formats 10-33
Extended Formats 10-35
SVC_REQ 20 Examples 10-37
SVC_REQ 21: User-Defined Fault Logging 10-38
SVC_REQ 22: Mask/Unmask Timed Interrupts 10-40
SVC_REQ 23: Read Master Checksum 10-41
SVC_REQ 24: Reset Module 10-42
SVC_REQ 25: Disable/Enable EXE Block and Standalone C Program Checksums .. 10-43
SVC_REQ 29: Read Elapsed Power Down Time 10-44
SVC_REQ 32: Suspend/Resume I/O Interrupt 10-45
SVC_REQ 45: Skip Next I/O Scan 10-47
SVC_REQ 50: Read Elapsed Time Clock 10-48
SVC_REQ 51: Read Sweep Time from Beginning of Sweep 10-50

PID Built-in Function Block 11-1

Operands of the PID Function 11-2
Operands for LD Version of PID Function Block 11-2
Operands for FBD Version of PID Function Block 11-3
Reference Array for the PID Function 11-4
Scaling Input and Outputs 11-4
Reference Array Parameters 11-5
Operation of the PID Function 11-10
Automatic Operation 11-10
Manual Operation 11-10
Time Interval for the PID Function 11-11
PID Algorithm Selection (PIDISA or PIDIND) and Gain Calculations 11-12

Error Term 11-13
Derivative Term 11-13
CV Bias Term 11-14
CV Amplitude and Rate Limits 11-14
Sample Period and PID Function Block Scheduling 11-15
Determining the Process Characteristics 11-16
Setting Tuning Loop Gains 11-17
Basic Iterative Tuning Approach 11-17
Setting Loop Gains Using the Ziegler and Nichols Tuning Approach 11-17
Ideal Tuning Method 11-18
Example 11-19

Structured Text Programming 12-1

Language Overview 12-1
Statements 12-1
Expressions 12-1
Operators 12-2
Structured Text Syntax 12-3
Statement Types 12-4
Assignment Statement 12-5
Function Call 12-6
RETURN Statement 12-10
IF Statement 12-11
CASE Statement 12-12
FOR Statement 12-14
WHILE Statement 12-16
REPEAT Statement 12-17
ARG_PRES Statement 12-18
Exit Statement 12-19

Communications 13-1

Ethernet Communications 13-2
Embedded Ethernet Interface 13-2
Ethernet Interface Modules 13-2
Serial Communications 13-3
Serial Port Communications Capabilities 13-3
Configurable Stop Mode Protocols 13-4
Serial Port Pin Assignments 13-4
Serial Port Baud Rates 13-7
Series 90-70 Communications and Intelligent Option Modules 13-8
Communications Coprocessor Module (CMM) 13-8
Programmable Coprocessor Module (PCM) 13-9
DLAN/DLAN+ (Drives Local Area Network) Interface 13-10

Serial I/O, SNP and RTU Protocols 14-1

Configuring Serial Ports Using the COMM_REQ Function 14-2
COMM_REQ Function Example 14-2
Timing 14-2
Sending Another COMM_REQ to the Same Port 14-2
Invalid Port Configuration Combinations 14-3
COMM_REQ Command Block Parameter Values 14-3
Sample COMM_REQ Command Blocks 14-4
Calling Serial I/O COMM_REQs from the CPU Sweep 14-7
Compatibility 14-7
Status Word for Serial I/O COMM_REQs 14-8
Serial I/O COMM_REQ Commands 14-9
Overlapping COMM_REQs 14-9
Initialize Port Function (4300) 14-10
Set Up Input Buffer Function (4301) 14-11
Flush Input Buffer Function (4302) 14-11
Read Port Status Function (4303) 14-12
Write Port Control Function (4304) 14-14
Cancel COMM_REQ Function (4399) 14-15
Autodial Function (4400) 14-16
Write Bytes Function (4401) 14-18
Read Bytes Function (4402) 14-19
Read String Function (4403) 14-21
RTU Slave Protocol 14-23
Message Format 14-23
Cyclic Redundancy Check (CRC) 14-28
Calculating the CRC-16 14-29
Sample CRC-16 Calculation 14-29
Calculating the Length of Frame 14-31
RTU Message Descriptions 14-32
RTU Scratch Pad 14-48
Communication Errors 14-49
RTU Slave/SNP Slave Operation With Programmer Attached 14-51
SNP Slave Protocol 14-52
Permanent Datagrams 14-52
Communication Requests (COMM_REQs) for SNP 14-52

Fault Handling 15-1

Overview 15-2
System Response to Faults 15-2
Fault Tables 15-2
Fault Actions and Fault Action Configuration 15-3
Using the Fault Tables 15-4
Controller Fault Table 15-4
I/O Fault Table 15-6

System Handling of Faults 15-8
System Fault References 15-8
Using Fault Contacts 15-11
Using Point Faults 15-13
Using Alarm Contacts 15-13
Controller Fault Descriptions and Corrective Actions 15-14
Loss of or Missing Rack (Group 1) 15-15
Loss of or Missing Option Module (Group 4) 15-16
Addition of, or Extra Rack (Group 5) 15-16
Reset of, Addition of, or Extra Option Module (Group 8) 15-17
System Configuration Mismatch (Group 11) 15-18
System Bus Error (Group 12) 15-23
CPU Hardware Failure (Group 13) 15-24
Module Hardware Failure (Group 14) 15-25
Option Module Software Failure (Group 16) 15-25
Program or Block Checksum Failure (Group 17) 15-27
Battery Status (Group 18) 15-28
Constant Sweep Time Exceeded (Group 19) 15-28
System Fault Table Full (Group 20) 15-29
I/O Fault Table Full (Group 21) 15-29
User Application Fault (Group 22) 15-29
CPU Over Temperature (Group 24) 15-31
Power Supply Fault (Group 25) 15-32
No User Program on Power-Up (Group 129) 15-32
Corrupted User Program on Power-Up (Group 130) 15-33
Window Completion Failure (Group 131) 15-34
Password Access Failure (Group 132) 15-34
Null System Configuration for Run Mode (Group 134) 15-34
CPU System Software Failure (Group 135) 15-35
Communications Failure During Store (Group 137) 15-36
Noncritical CPU Software Event (Group 140) 15-37
I/O Fault Descriptions and Corrective Actions 15-38
Fault Extra Data 15-38
I/O Fault Groups 15-38
I/O Fault Categories 15-39
Circuit Faults (Category 1) 15-41
Loss of Block (Category 2) 15-47
Addition of Block (Category 3) 15-48
I/O Bus Fault (Category 6) 15-49
Module Fault (Category 8) 15-50
Addition of IOC (Category 9) 15-51
Loss of or Missing IOC (Category 10) 15-51
IOC (I/O Controller) Software Fault (Category 11) 15-52
Forced and Unforced Circuit (Categories 12 and 13) 15-52
Loss of or Missing I/O Module (Category 14) 15-53
Addition of I/O Module (Category 15) 15-53

Extra I/O Module (Category 16) 15-53
Extra Block (Category 17) 15-54
IOC Hardware Failure (Category 18) 15-54
GBC Stopped Reporting Faults (Category 19) 15-54
GBC Software Exception (Category 21) 15-55
Block Switch (Category 22) 15-56
Reset of IOC (Category 27) 15-56

Performance Data A-1

Boolean Execution Times A-1
Instruction Timing A-2
Function Execution Times A-2
Incremental Times A-10
Overhead Sweep Impact Times A-12
Base Sweep Times A-12
What the Sweep Impact Tables Contain A-14
Programmer Sweep Impact Times A-14
I/O Scan and I/O Fault Sweep Impact A-15
Ethernet Global Data Sweep Impact A-21
Sweep Impact of Intelligent Option Modules A-24
I/O Interrupt Performance and Sweep Impact A-25
Timed Interrupt Performance A-26
Example of Predicted Sweep Time Calculation A-27

User Memory Allocation B-1

Items that Count Against User Memory B-2
User Program Memory Usage B-3
%L and %P Program Memory B-3
Program Logic and Overhead B-3

Converting Series 90 Applications to PACSystems C-1

PACSystems – Series 90 Comparison C-2
CPU Operation C-2
Logic C-3
Blocks C-8
PACSystems Functions C-9
Online Editing Mode C-13
Variables C-14
Communications C-15
Ethernet Global Data C-16
Flash C-16
Memory C-17
Redundancy C-18
Genius C-19
I/O and Intelligent Modules C-20

Converting an Application from Series 90-70 to PACSystems....C-22
Preparing for the Conversion....C-22
Converting the Target....C-24
Changes Made During the 90-70 to PACSystems RX7i Conversion....C-25
Finishing the Conversion....C-26
Converting an Application from Series 90-30 to PACSystems....C-28
Preparing for the Conversion....C-28
Converting the Target....C-30
Changes Made During the 90-30 to PACSystems RX3i Conversion....C-31
Finishing the Conversion....C-32

Chapter 1 Introduction

This manual contains general information about PACSystems CPU operation and program content. It also provides detailed descriptions of specific programming requirements.

Chapter 1 provides a **general introduction** to the PACSystems family of products, including new features, product overviews, and a list of related documentation.

CPU hardware features and specifications are provided in chapter 2.

Installation procedures are described in the *PACSystems RX7i Installation Manual,* GFK-2223 and the *PACSystems RX3i Installation Manual,* GFK-2314.

CPU Configuration is described in chapter 3. Configuration using the programming software determines characteristics of module operation and establishes the program references used by each module in the system.

Ethernet Configuration for the embedded RX7i Ethernet interface is described in chapter 4. (For details on PACSystems Ethernet communications and configuration of the RX7i and RX3i Ethernet Interface modules, refer to *TCP/IP Ethernet Communications for PACSystems*, GFK-2224.)

CPU Operation is described in chapter 5.

Programming Features are described in chapters 6 through 10 and Appendix A.

- Elements of an Application Program: chapter 6
- Program Data: chapter 7
- Ladder Diagram (LD) instruction set reference: chapter 8
- Function Block Diagram (FBD) instruction set reference: chapter 9
- The Service Request Function: chapter 10
- The PID Function: chapter 11
- Structured Text (ST): chapter 12

Ethernet and Serial Communications are described in chapter 13.

Serial I/O, SNP, and RTU Protocols are described in chapter 14.

Fault Handling is described in chapter 15.

Instruction Timing is provided in appendix A.

User Memory Allocation is described in Appendix B.

Converting Applications from Series 90 to PACSystems is discussed in appendix C, which also summarizes operational differences between the PACSystems and Series 90 control systems.

New Features

Note: A given feature may not be implemented on all PACSystems CPUs. To determine whether a feature is available on a given CPU model and firmware version, please refer to the *Important Product Information* (IPI) document provided with the CPU.

This revision of the PACSystems CPU Reference Manual includes the following new features:

LREAL data type – PACSystems Controllers now support the use of 64-bit double-precision floating point LD, ST, and FBD logic.

SCAN_SET_IO function – Scans the I/O of a specified scan set, which is defined in hardware configuration. You can choose to scan the inputs, the outputs, or both inputs and outputs of the specified scan set.

Dynamic Ethernet Global Data (Run mode store of EGD) – Ability to perform a run mode store of EGD allows you to add, delete or modify EGD exchanges without stopping the controller. For details on using this feature, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.

Dynamic redundancy transfer lists (Run mode store of transfer lists) – Allows you to use a run mode store to modify the transfer list in a redundancy system without stopping the controllers. For details on using this feature, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308.

PACSystems Control System Overview

The PACSystems controller environment combines performance, productivity, openness and flexibility. The PACSystems control system integrates advanced technology with GE Fanuc's existing systems. The result is seamless migration that protects your investment in I/O and application development.

Programming and Configuration

Proficy® Machine Edition programming software provides a universal engineering development environment for all programming, configuration and diagnostics of PACSystems. A PACSystems CPU is programmed and configured using the programming software to perform real time control of machines, processes, and material handling systems. The CPU communicates with I/O and smart option modules through a rack-mounted backplane. It communicates with the programmer and/or HMI devices via the Ethernet ports (may be embedded for RX7i) or via the serial ports 1 and 2 using GE Fanuc Serial I/O, or Modbus RTU slave protocols.

Process Systems

PACSystems CPUs with firmware version 5.0 and later support Proficy Process Systems (PPS). PPS is a complete, tightly integrated, seamless process control system using PACSystems, Proficy HMI/SCADA, and Proficy Production Management Software to provide control, optimization, and performance management to manage and monitor batch or continuous manufacturing. It delivers the tools required to design, implement, document, and maintain an automated process. For information about purchasing PPS software, refer to http://www.gefanuc.com/.

PACSystems CPU Models

Family	Catalog Number	Description
RX3i CPUs	IC695CPU310	300MHz Celeron CPU, 10 MB user memory
	IC695CPU320	1 GHz Celeron-M CPU, 64 MB user memory
	IC695NIU001	300MHz Celeron NIU. For information, see the *PACSystems RX3i Ethernet NIU User's Manual,* GFK-2439
RX7i CPUs with embedded Ethernet Interface	IC698CPE010	300MHz, Celeron CPU, 10MB user memory
	IC698CPE020	700MHz, Pentium CPU, 10 MB user memory,
	IC698CPE030	600MHz, Pentium-M CPU, 64MB user memory
	IC698CPE040	1800MHz, Pentium-M CPU, 64MB user memory
RX7i Redundancy CPUs with embedded Ethernet Interface	IC698CRE020	700MHz, Pentium CPU, 10 MB user memory
	IC698CRE030	600MHz, Pentium-M CPU, 64MB user memory
	IC698CRE040	1800MHz, Pentium-M CPU, 64MB user memory

PACSystems CPU models have the following features in common:

- Programming in Ladder Diagram, Function Block Diagram, Structured Text and C.
- Floating point (real) data functions.
- Configurable data and program memory.
- Battery-backed RAM for user data (program, configuration, register data, and symbolic variable) storage
- Non-volatile built-in flash memory for user data (program, configuration, register data, and symbolic variable) storage. Use of this flash memory is optional.
- Battery backup for program, data, and time of day clock.
- Configurable Run/Stop mode switch.
- Embedded RS-232 and RS-485 communications.
- Up to 512 program blocks. Maximum size for a block is 128KB.
- Auto Located Symbolic Variables, which allows you to create a variable without specifying a reference address.
- Bulk memory area accessed via reference table %W. The upper limit of this memory area can be configured to the maximum available user RAM.
- Larger reference table sizes, compared to Series 90 CPUs: 32Kbits for discrete %I and %Q and up to 32K words each for analog %AI and %AQ.
- Online Editing mode that allows you to easily test modifications to a running program. (For details on using this feature, refer to the programming software online help and *Proficy™ Logic Developer Getting Started,* GFK-1918.)
- Bit in word referencing that allows you to specify individual bits in a WORD reference in retentive memory as inputs and outputs of Boolean expressions, function blocks, and calls that accept bit parameters.
- In-system upgradeable firmware.

RX3i Overview

The RX3i control system hardware consists of an RX3i universal backplane and up to seven Series 90-30 expansion or remote racks. The CPU can be in any slot in the universal backplane except the last slot, which is reserved for the serial bus transmitter, IC695LRE001.

The RX3i supports user defined Function Blocks (LD logic only) and Structured Text programming.

The RX3i universal backplane uses a dual bus that provides both:

- High-speed, PCI for fast throughput of new advanced I/O.
- Serial backplane for easy migration of existing Series 90-30 I/O

The RX3i universal backplane and Series 90-30 expansion/remote racks support the Series 90-30 Genius Bus Controller and Motion Control modules, and most Series 90-30/RX3i discrete and analog I/O with catalog prefixes IC693 and IC694. RX3i modules with catalog prefixes IC695, including the Ethernet and other communications modules can only be installed in the universal backplane. See the *PACSystems RX3i System Manual,* GFK-2314 for a list of supported modules.

RX3i communications features include:

- Open communications support includes Ethernet, and serial protocols. The Ethernet Interface (resides in a backplane slot) has dual RJ-45 ports connected through an auto-sensing switch. This eliminates the need for rack-to-rack switches or hubs. The Ethernet Interface supports upload, download and online monitoring, and provides 32 SRTP channels and allows a maximum of 48 simultaneous SRTP server connections. For details on Ethernet Interface capabilities, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.
- The RX3i supports PROFIBUS communications via the PROFIBUS Master module. For details, refer to the *PACSystems RX3i PROFIBUS Modules User's Manual,* GFK-2301.
- Two serial ports, one RS-232 and one RS-485.

RX7i Overview

The RX7i control system hardware consists of an RX7i rack and up to seven Series 90-70 expansion racks. The CPU resides in slot 1 of the main rack. RX7i racks use a VME64 backplane that provides up to four times the bandwidth of existing VME based systems, including the current Series 90-70 systems for faster I/O throughput. The VME64 base supports all standard VME modules including Series 90-70 I/O and VMIC modules.

Expansion racks support Series 90-70 discrete and analog I/O, the Genius Bus Controller, and the High Speed Counter. The CPU provides an embedded auto-sensing 10/100 Mbps half/full duplex Ethernet interface.

RX7i supports hot standby (HSB) CPU redundancy, which allows a critical application or process to continue operating if a failure occurs in any single component. A CPU redundancy system consists of an active unit that actively controls the process and a backup unit that is synchronized with the active unit and can take over the process should it become necessary. Each unit must have a redundancy CPU, (IC698CRE020, CRE030 or CRE040). The redundancy communication path is provided by

IC698RMX016 Redundancy Memory Xchange (RMX) modules set up as redundancy links. For details on the operation of an RX7i redundancy system, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308.

Note: Extended operation with dissimilar CPU types is *not allowed.* During normal operation, the primary and secondary units in an HSB redundancy system must have the same CPU model type.

The primary and secondary units of an HSB redundancy system can have dissimilar model types for a limited time, for the purpose of system upgrade only. Fail wait times for the higher performance CPU in a dissimilar redundant pair may need to be increased to allow synchronization.

RX7i communications features include:

- Open communications support includes Ethernet, Genius, and serial protocols.
- A built-in 10/100mb Ethernet interface that has dual RJ-45 ports connected through an auto-sensing switch for upload, download and online monitoring. This eliminates the need for rack-to-rack switches or hubs. The CPU Ethernet Interface provides basic remote control system monitoring from a web browser and allows a combined total of up to 16 web server and FTP connections. For details on Ethernet Interface capabilities, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.
- Two serial ports, one RS-232 and one RS-485.
- An RS-232 isolated Ethernet station manager serial port.

Migrating Series 90 Applications to PACSystems

The PACSystems control system provides cost-effective expansion of existing systems. Support for existing Series 90 modules, expansion racks and remote racks protects your hardware investment. You can upgrade on your timetable without disturbing panel wiring.

- The RX3i supports most Series 90-30 modules, expansion racks, and remote racks. For a list of supported I/O, Communications, Motion, and Intelligent modules, see the *PACSystems RX3i Installation Manual,* GFK-2314.
- The RX7i supports most existing Series 90-70 modules, expansion racks and Genius networks. For a list of supported I/O, Communications, and Intelligent modules, see the *PACSystems RX7i Installation Manual,* GFK-2223.
- Conversion of Series 90-70 and Series 90-30 programs preserves existing development effort.
- Conversion of VersaPro and Logicmaster applications to Machine Edition allows smooth transition to PACSystems.

PACSystems Documentation

PACSystems Manuals

PACSystems CPU Reference Manual, GFK-2222
TCP/IP Ethernet Communications for PACSystems, GFK-2224
Station Manager for PACSystems, GFK-2225
PACSystems C Toolkit User's Guide, GFK-2259
PACSystems Memory Xchange Modules User's Manual, GFK-2300
PACSystems Hot Standby CPU Redundancy User's Manual, GFK-2308
Proficy Machine Edition Logic Developer Getting Started, GFK-1918
Proficy Process Systems Getting Started Guide, GFK-2487

RX3i Manuals

PACSystems RX3i Hardware and Installation Manual, GFK-2314
DSM324i Motion Controller for PACSystems RX3i and Series 90-30, GFK-2347
PACSystems RX3i PROFIBUS Modules User's Manual, GFK-2301
PACSystems RX3i MAXON Software User's Manual, GFK-2409
PACSystems RX3i Ethernet NIU User's Manual, GFK-2439

RX7i Manuals

PACSystems RX7i Hardware and Installation Manual, GFK-2223
PACSystems RX7i User's Guide to Integration of VME Modules, GFK-2235
Genius Bus Controller User's Manual, GFK-2017

Series 90 Manuals

Series 90 Programmable Coprocessor Module and Support Software, GFK-0255
Series 90 PLC Serial Communications Driver User's Manual, GFK-0582
C Programmer's Toolkit for Series 90 PLCs User's Manual, GFK-0646
Installation Requirements for Conformance to Standards, GFK-1179

TCP/IP Ethernet Communications for the Series 90 PLC Station Manager Manual, GFK-1186

Series 90-70 Programmable Controller Installation Manual, GFK-0262
Series 90-70 CPU Instruction Set Reference Manual, GFK-0265
Series 90-30 Genius Bus Controller, GFK-1034
Series 90-30 System Manual, GFK-1411

Ethernet NIU User's Manual, GFK-2296
Genius I/O System User's Manual, GEK-90486-1
Genius I/O Analog and Discrete Blocks User's Manual, GEK-90486-2

In addition to these manuals, datasheets and product update documents describe individual modules and product revisions. The most recent PACSystems documentation is available on the GE Fanuc website: http://www.gefanuc.com/.

Technical Support

If you purchased this product through a GE Fanuc Authorized Channel Partner, please contact them directly.

General Contact Information

Online Technical Support and GlobalCare: http://www.gefanuc.com/support

Additional information: http://www.gefanuc.com/

Technical Support

If you have technical problems that cannot be resolved with the information in this guide, please contact us by telephone or email, or on the web at http://www.gefanuc.com/support.

Americas

Online Technical Support: http://www.gefanuc.com/support

Phone: 1-800-GE FANUC (1-800-433-2682)

International Americas Direct Dial: 1-434-978-5100

Technical Support Email: mailto:support@gefanuc.com

Customer Care Email: mailto:customer.care@gefanuc.com

Primary language of support: English

Europe, the Middle East, and Africa

Online Technical Support: http://www.gefanuc.com/support

Phone: +800 1 GE FANUC (+800-1-433-2682)

Technical Support Email: mailto:support.emea@gefanuc.com

Customer Care Email: mailto:customercare.emea@gefanuc.com

Primary languages of support: English, French, German, Italian, Czech

Asia Pacific

Online Technical Support: http://www.gefanuc.com/support

Phone: +86-400-820-8208

+86-21-3217-4826 (India, Indonesia, and Pakistan)

Technical Support Email: mailto:support.cn@gefanuc.com (China)

mailto:support.jp@gefanuc.com (Japan)

mailto:support.in@gefanuc.com (remaining Asia customers)

Customer Care Email: mailto:customercare.apo@gefanuc.com

Chapter 2 CPU Features and Specifications

This chapter provides details on the hardware features of the PACSystems CPUs and their specifications.

Common CPU Features

Firmware Storage in Flash Memory

The CPU uses non-volatile flash memory for storing the operating system firmware. This allows firmware to be updated without disassembling the module or replacing EPROMs. The operating system firmware is updated by connecting a PC compatible computer to the module's serial port and running the software included with the firmware upgrade kit.

Operation, Protection, and Module Status

Operation of the CPU can be controlled by the three-position Run/Stop switch or remotely by an attached programmer and programming software. Program and configuration data can be locked through software passwords. The status of the CPU is indicated by the CPU LEDs on the front of the module. (On the RX7i CPUs, seven LEDs indicate the status of the Ethernet interface.) For details, see "Indicators" for each PACSystems family.

Note: The RESET pushbutton is provided to support future features and has no effect on CPU operation in the current version.

Ethernet Global Data

Each PACSystems CPU supports up to 255 simultaneous EGD pages across all Ethernet interfaces in the PLC. EGD pages must be configured in the programming software and stored into the CPU. The EGD configuration can also be loaded from the CPU into the programming software. Both produced and consumed pages can be configured. PACSystems CPUs support the use of only part of a consumed EGD page, and EGD page production and consumption to the broadcast IP address of the local subnet.

The PACSystems CPU supports 2msec EGD page production and timeout resolution. EGD pages can be configured for a production period of 0, indicating the page is to be produced every output scan. The minimum period for these "as fast as possible" pages is 2msec.

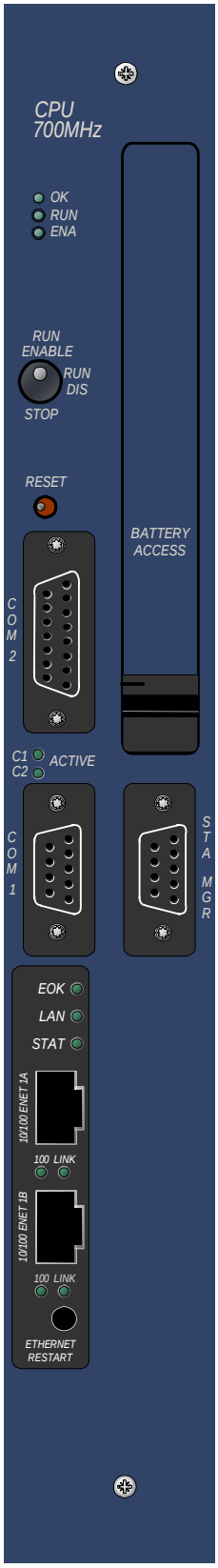
During EGD configuration, PACSystems Ethernet interfaces are identified by their Rack/Slot location.

RX7i Features and Specifications

CPE010, CPE020 and CRE020 Models

- **IC698CPE010:** 300MHz CPU microprocessor
- **IC698CPE020:** 700MHz CPU microprocessor
- **IC698CRE020:** 700MHz CPU microprocessor with redundancy

For details on the operation of the embedded Ethernet interface, refer to page 2-8.



CPU Serial Ports

The CPU has three independent, on-board serial ports, accessed by connectors on the front of the module. Ports 1 and 2 provide serial interfaces to external devices. Port 1 or port 2 can be used for firmware upgrades. The third on-board serial port is used as the Ethernet Station Manager port. All serial ports are isolated. For serial port pin assignments and details on serial communications, refer to chapter 14.

CPU Indicators

Five CPU LEDs indicate the operating status of various CPU functions.

LED State ● On ✱ Blinking ○ Off			***CPU Operating State***
●	**OK**	On	CPU has passed its powerup diagnostics and is functioning properly.
○	**OK**	Off	CPU problem. EN and RUN LEDs may be blinking in an error code pattern, which can be used by technical support for troubleshooting. This condition and any error codes should be reported to your technical support representative.
✱	**OK**, **EN**, **RUN** Blinking in unison		CPU is in boot mode and is waiting for a firmware update through a serial port.
✱	**OK** Blinking Other LEDs off.		CPU in Stop/Halt state; possible watchdog timer fault. Refer to the fault tables. If the programmer cannot connect, cycle power with battery attached and refer to fault tables.
●	**RUN**	On	CPU is in Run mode
○	**RUN**	Off	CPU is in Stop mode.
●	**EN**	On	Output scan is enabled.
○	**EN**	Off	Output scan is disabled.
✱	**C1** (port 1) **C2** (port 2)	Blinking Blinking	Signal activity on port.

Specifications – CPE010, CPE020 and CRE020 Models

For environmental specifications, see "RX7i General Specifications" in Appendix A of the *RX7i Installation Manual,* GFK-2223.

Battery Life: Memory retention	For estimated battery life under various conditions, refer to page 2-4.
Program storage	Up to 10 Mbytes of battery-backed RAM 10 Mbytes of non-volatile flash user memory
Power requirements CPE010 CPE020, CRE020	 +5 VDC: 3.2 Amps nominal +12 VDC: 0.042 Amps nominal -12 VDC: 0.008 Amps nominal +5 VDC: 4.5 Amps nominal +12 VDC: 0.042 Amps nominal -12 VDC: 0.008 Amps nominal
Operating Temperature	CPE010: 0 to 50°C (32°F to 122°F 0 to 60°C (32°F to 140°F) with fan tray CPE020, CRE020: 0 to 60°C (32°F to 140°F), fan tray required
Floating point	Yes
Boolean execution speed, typical CPE010 CPE020, CRE020	 0.195ms per 1000 Boolean contacts/coils 0.14ms per 1000 Boolean contacts/coils
Time of Day Clock accuracy Elapsed Time Clock (internal timing) accuracy	Maximum drift of ±9 seconds per day. Can be synchronized to an Ethernet time master within ±2ms of the SNTP time stamp. ±0.01% maximum
Embedded communications	RS-232, RS-485, Ethernet interface
Serial Protocols supported	Modbus RTU Slave, SNP, Serial I/O To determine availability for a given firmware version, please refer to the *Important Product Information* document provided with the CPU.
Ethernet Ports	Embedded auto-sensing 10/100 Mbps half/full duplex Ethernet interface
VME Compatibility	System designed to support the VME64 standard ANSI/VITA 1
Program blocks	Up to 512 program blocks. Maximum size for a block is 128KB.
Memory (For a detailed listing of memory areas, refer to chapter 7.)	*%I and %Q:* 32Kbits for discrete *%AI and %AQ:* configurable up to 32Kwords *%W:* configurable up to the maximum available user RAM Managed memory (*Symbolic and I/O variables combined):* configurable up to 10 Mbytes
Error Checking and Correction	CRE020 only.

Ethernet Interface Specifications	
Web-based data monitoring	Up to 16 web server and FTP connections (combined)
Ethernet data rate	10Mb/sec and 100Mb/sec
Physical interface	10BaseT RJ45
WinLoader support	Yes
Number of EGD configuration-based pages	255
Time synchronization	SNTP
Selective consumption of EGD	Yes
Load EGD configuration from PLC to programmer	Yes
Remote Station Manager over UDP	Yes
Local Station Manager (RS-232)	Dedicated RS-232 port
Configurable Advanced User Parameters	Yes

Battery Life Estimates for CPE010 and CPE020/CRE020

To avoid loss of RAM memory contents, routine maintenance procedures should include scheduled replacement of the CPU's lithium battery pack, IC698ACC701. The following table lists estimates of battery life that can be used to develop a battery replacement schedule.

Nominal IC698ACC701 Battery Pack Installed Life

Controller	***Average Temperature***	***Nominal Life with Applied Power On:***	
		100% of the Time	***0% of the Time***
IC698CPE010 IC698CPE020 IC698CRE020	20°C (68°F)	5 years	40 days

The IC698ACC701 battery pack has a nominal shelf life of 5 years when stored at an average temperature of 20°C (68°F).

Note: An external Auxiliary Battery Module, IC693ACC302 can be used to provide long-term battery backup for any PACSystems CPU. For details, refer to the datasheet for the Auxiliary Battery Module, GFK-2124.

Error Checking and Correction, IC698CRE020

Redundancy CPUs are shipped with error checking and correction (ECC) enabled. Enabling ECC results in slightly slower system performance, primarily during power-up, because it uses an extra 8 bits that must be initialized. If you upgrade the firmware on the non-redundancy CPU model IC698CPE020 to support redundancy, you must set the ECC jumper to the enabled state as described in the installation instructions provided with the upgrade kit.

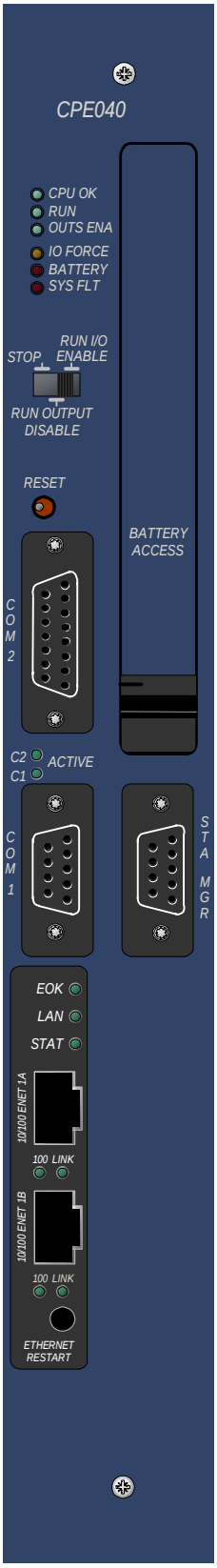
The CRE020 performance measurements provided in appendix A were done with ECC enabled.

For details on ECC, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308.

CPE030/CRE030 and CPE040/CRE040 Models

- CPE030/CRE030: 600MHz Pentium-M microprocessor
- CPE040/CRE040: 1800MHz Pentium-M microprocessor

For details on the embedded Ethernet interface, refer to page 2-8.



CPU Serial Ports

The CPU has three independent, isolated, on-board serial ports, accessed by connectors on the front of the module. Ports 1 and 2 provide serial interfaces to external devices and can be used for firmware upgrades. The third serial port is a dedicated Ethernet Station Manager port. For serial communications, see chapter 14.

CPU Indicators

Seven CPU LEDs indicate CPU operating status.

LED State (● On ✱ Blinking ○ Off)			***CPU Operating State***
●	**CPU OK**	On	CPU has passed its powerup diagnostics and is functioning properly.
○	**CPU OK**	Off	CPU problem. RUN and OUTPUTS ENABLED LEDs may be blinking in an error code pattern, which can be used by technical support for diagnostics. This condition and any error codes should be reported to your technical support representative.
✱	**CPU OK**, **OUTS ENA, RUN**	Blinking in unison	CPU is in boot mode and is waiting for a firmware update through a serial port.
✱	**OK** Other LEDs off.	Blinking	CPU in Stop/Halt state; possible watchdog timer fault. Refer to the fault tables. If the programmer cannot connect, cycle power with battery attached and refer to fault tables.
○	**RUN**	Off	CPU is in Stop mode.
●	**OUTS ENA**	On	Output scan is enabled.
○	**OUTS ENA**	Off	Output scan is disabled.
●	**I/O FORCE**	On	Override is active on a bit reference (Not used by CRE030 and CRE040.)
●	**BATTERY**	On	Battery has failed or is not attached. To provide reliable backup, routine maintenance should include scheduled battery replacement. See “Battery Life Estimates” provided on page 2-7.
●	**SYS FAULT**	On	CPU is in Stop/Faulted mode because a fatal fault has occurred.
✱	**C1** (port 1) **C2** (port 2)	Blinking Blinking	Signal activity on port.

Specifications – CPE030/CRE030 and CPE040/CRE040 Models

For environmental specifications, see "RX7i General Specifications" in Appendix A of the *RX7i Installation Manual,* GFK-2223.

Battery: Memory retention	Uses an IC693ACC302 Auxiliary Battery Module. For estimated battery life under various conditions, see page 2-7. For details on the operation of the Auxiliary Battery Module, refer to the datasheet, GFK-2124. **Note:** The IC698ACC701 RX7i Replacement Battery is ***not compatible*** with the CPE030, CRE030, CPE040 and CRE040 CPU modules.
Program storage	Up to 64 Mbytes of battery-backed RAM 64 Mbytes of non-volatile flash user memory
Power requirements	CPE030/CRE030:+5 VDC: 3.2 Amps nominal +12 VDC: 0.003 Amps nominal -12 VDC: 0.003 Amps nominal CPE040/CRE040: +5 VDC: 6.8 Amps nominal +12 VDC: 0.003 Amps nominal -12 VDC: 0.003 Amps nominal
Operating temperature	CPE030/CRE030: 0 to 50°C (32°F to 122°F 0 to 60°C (32°F to 140°F) with fan tray CPE040/CRE040: 0 to 60°C (32°F to 140°F), fan tray required
Floating point	Yes
Boolean execution speed, typical:	CPE030/CRE030: 0.069ms per 1000 Boolean contacts/coils CPE040/CRE040: 0.024ms per 1000 Boolean contacts/coils
Time of Day Clock accuracy	Maximum drift of ±2 seconds per day. Can be synchronized to an Ethernet time master within ±2ms of the SNTP time stamp.
Elapsed Time Clock (internal timing) accuracy	±0.01% maximum
Embedded communications	RS-232, RS-485, Ethernet interface
Serial Protocols supported	Modbus RTU Slave, SNP, Serial I/O To determine availability for a given firmware version, please refer to the *Important Product Information* document provided with the CPU.
Ethernet Ports	Embedded auto-sensing 10/100 Mbps half/full duplex Ethernet interface
[Optional] Station Manager cable for Ethernet Interface	IC200CBL001
VME Compatibility	System designed to support the VME64 standard ANSI/VITA 1
Program blocks	Up to 512 program blocks. Maximum size for a block is 128KB.
Memory (For a detailed listing of memory areas, refer to chapter 7.)	*%I and %Q:* 32Kbits for discrete *%AI and %AQ:* configurable up to 32Kwords *%W:* configurable up to the maximum available user RAM Managed memory (*Symbolic and I/O variables combined):* configurable up to 10 Mbytes
Error checking and correction	CRE030 and CRE040 only

Ethernet Interface Specifications	
Web-based data monitoring	Up to 16 web server and FTP connections (combined)
Ethernet data rate	10Mb/sec and 100Mb/sec
Physical interface	10BaseT RJ45
WinLoader support	Yes
Number of EGD configuration-based pages	255
Time synchronization	SNTP
Selective consumption of EGD	Yes
Load EGD configuration from PLC to programmer	Yes
Remote Station Manager over UDP	Yes
Local Station Manager (RS-232)	Dedicated RS-232 port
Configurable Advanced User Parameters	Yes

Battery Life Estimates for IC698CPE030/CRE030 and IC698CPE040/CRE040

To avoid loss of RAM memory contents, routine maintenance procedures should include scheduled replacement of the IC693ACC302 battery pack. The following table lists estimates of battery life that can be used to develop a battery replacement schedule.

Nominal Battery Life with Applied Power OFF (100% Usage)

0ºC*	***20ºC***	***60ºC****
20 days	30 days	35 days

* The nominal backup values are estimated at 20ºC. Backup time increases approximately 17% at 60ºC and decreases approximately 32% at 0ºC.

Error Checking and Correction, IC698CRE030 and IC698CRE040

Redundancy CPUs are shipped with error checking and correction (ECC) enabled. Enabling ECC results in slightly slower system performance, primarily during power-up, because it uses an extra 8 bits that must be initialized. If you upgrade the firmware on a non-redundancy CPU model to support redundancy, you must set the ECC jumper to the enabled state as described in the installation instructions provided with the upgrade kit.

For details on ECC, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308.

Note: Multiple Recoverable Memory Error faults may be generated when a single-bit ECC error is detected. When a single-bit ECC error is detected, the value presented to the microprocessor is corrected. However, the value stored in RAM is not corrected until the next time the microprocessor writes to that RAM location.

Embedded Ethernet Interface

Ethernet Ports

The embedded Ethernet Interface provides two RJ-45 Ethernet ports. Either or both of these ports may be connected to other Ethernet devices. Each port automatically senses the data rate (10Mbps or 100Mbps), duplex (half duplex or full duplex), and cabling arrangement (straight through or crossover) of the attached link.) For Ethernet port pin assignments, refer to chapter 12. For details on Ethernet communications, refer to the following manuals:

TCP/IP Ethernet Communications for PACSystems User's Guide, GFK-2224
PACSystems TCP/IP Communications Station Manager Manual, GFK-2225

Caution

The two ports on the Ethernet Interface must not be connected, directly or indirectly to the same device. The hub or switch connections in an Ethernet network must form a tree; otherwise duplication of packets may result.

Ethernet Interface Indicators

The Ethernet Interface indicators consist of seven light emitting diodes (LEDs). All are single-color green LEDs controlled by the Ethernet interface.

- Module OK (EOK)
- LAN online (LAN)
- Status (STAT)
- Two activity LEDS (LINK)
- Two speed LEDS (100)

The **EOK**, **LAN**, and **STAT** LEDs are grouped together and indicate the state and status of the Ethernet interface.

Each Ethernet port has two green LED indicators, **Link** and **100**. The **LINK** LED indicates the network link status and activity. This LED is illuminated when the link is physically connected and blinks when traffic is detected at the port. Note that traffic at the port does not necessarily mean that traffic is present at the Ethernet interface, since the traffic may be going between ports of the switch. The **100** LED indicates the network data speed (10 or 100 Mb/sec). This LED is illuminated if the network connection is 100 Mbps.

LED operation is described in the following tables.

Ethernet LED Operation

LED State ● On ✹ Blinking ○ Off			***Ethernet Operating State***
✹	EOK	Blink error code	Hardware Failure
○	LAN	Off	
○	STAT	Off	
✹	EOK	Fast Blink	Performing Diagnostics
○	LAN	Off	
○	STAT	Off	
✹	EOK	Slow Blink	Waiting for Ethernet configuration from CPU
○	LAN	Off	
○	STAT	Off	
✹	EOK	Slow Blink*	Waiting for IP Address
● ✹ ○	LAN	On/Traffic/Off	
✹	STAT	Slow Blink*	
	(* *EOK and STAT blink in unison*)		
● ✹ ○	EOK	On	Operational
	LAN	On/Traffic/Off	
	STAT	On/Off	
✹	EOK	Slow Blink**	Software Load
✹	LAN	Slow Blink**	
✹	STAT	Slow Blink**	
	(** *All LEDs blink in unison*)		

EOK LED Operation

The EOK LED indicates whether the Ethernet interface is able to perform normal operation. This LED is on for normal operation and flashing for all other operations. When a hardware or unrecoverable runtime failure occurs, the EOK LED blinks a two-digit error code identifying the failure. The LED first blinks to indicate the most significant error digit, then after a brief pause blinks again to indicate the least significant error digit. After a long pause the error code display repeats.

EOK LED Blink Codes for Ethernet Hardware Failures

Blink Code	Description
0x12	Undefined or Unexpected Interrupt.
0x13	Timer failure during power up diagnostics.
0x14	DMA failure during power up diagnostics.
0x21	RAM failure during power up diagnostics.
0x22	Stack error during power up diagnostics.
0x23	Shared Memory Interface error during power up diagnostics.
0x24	Firmware CRC (cyclic redundancy check) error during power up or Factory Test.*
0x25	Run time exception
0x31	Undefined instruction or divide by zero
0x32	Software interrupt
0x33	Instruction prefetch abort
0x34	Data abort
0x35	Unexpected Runtime IRQ
0x36	Unexpected Runtime FIQ (fast interrupt request)
0x37	Reserved Exception or branch through zero

* CRC error or software error during normal operation causes Ethernet restart.

LAN LED Operation

The LAN LED indicates access to the Ethernet network. During normal operation and while waiting for an IP address, the LAN LED blinks to indicate network activity. This LED remains on when the Ethernet interface is not actively accessing the network but the network is available, and it is off if network access is not available. The definition of the network being available as indicated by this LED is that the Ethernet physical interface is available and one or both of the Ethernet ports is connected to an active network.

STAT LED Operation

The STAT LED indicates the condition of the Ethernet interface in normal operational mode. If the STAT LED is off, an event has been entered into the exception log and is available for viewing via the Station Manager interface. The STAT LED is on during normal operation when no events are logged.

In the other states, the STAT LED is either off or blinking and helps define the operational state of the module.

Ethernet Port LEDs Operation (100Mb and Link/Activity)

Each of the two Ethernet ports has two green LED indicators, **100** and **LINK**. The **100** LED indicates the network data speed (10 or 100 Mb/sec). This LED is illuminated if the network connection is 100 Mbps.

The **LINK** LED indicates the network link status and activity. This LED is illuminated when the link is physically connected and blinks when traffic is detected at the port. Note that traffic at the port does not necessarily mean that traffic is present at the Ethernet interface, since the traffic may be going between ports of the switch.

Ethernet Restart Pushbutton

The Ethernet Restart pushbutton is used to manually restart the Ethernet firmware without power cycling the entire control system. It is recessed to prevent accidental operation. The restart does not occur until the pushbutton is released.

The type of restart behavior is selected by the length of time that the pushbutton is depressed. The pushbutton-controlled restart operations are listed in the following table, along with the LED indications for each. In all cases, the EOK, LAN and STAT LEDs briefly turn on in unison as an LED test. The Ethernet port LEDs are not affected by a manual restart of the Ethernet firmware.

Restart Operation	***Depress Ethernet Restart pushbutton for***	***Ethernet LEDs Illuminated***
Normal restart	Less than 5 seconds	EOK, LAN, STAT
Restart without Ethernet plug-in applications	5 to 10 seconds	LAN, STAT
Restart into Firmware Update operation	More than 10 seconds	STAT

Normal Restart

When the Ethernet Restart pushbutton is pressed for less than 5 seconds, the Ethernet interface will restart into normal operation.

Restart without Ethernet plug-in applications

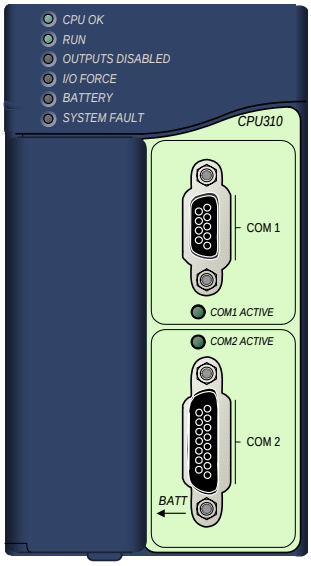
When the Restart pushbutton is pressed and held for 5 to 10 seconds, the Ethernet interface will restart into normal operation but does not start any optional Ethernet plug-in applications. This is typically done during troubleshooting.

Restart into with Firmware Update operation

When the Ethernet Restart pushbutton is pressed and held for more than 10 seconds, the Ethernet interface will restart into firmware update operation. This is typically done during troubleshooting to bypass possibly invalid firmware and allow valid firmware to be loaded using WinLoader.

Until the firmware update actually begins, you can manually exit the firmware update and restart with the existing firmware by pressing the Ethernet Restart pushbutton again.

RX3i Features and Specifications



IC695CPU310

- IC695CPU310: 300 MHz CPU microprocessor

Serial Ports

The CPU has two independent, on-board serial ports, accessed by connectors on the front of the module. Ports 1 and 2 provide serial interfaces to external devices. Either port can be used for firmware upgrades. For serial port pin assignments and details on serial communications, refer to chapter 12.

Indicators

The eight CPU LEDs indicate the operating status of various CPU functions.

LED State ● On ✱ Blinking ○ Off			***CPU Operating State***
●	**CPU OK**	On	CPU has passed its powerup diagnostics and is functioning properly.*
○	**CPU OK**	Off	CPU problem. RUN and OUTPUTS ENABLED LEDs may be blinking in an error code pattern, which can be used by technical support for troubleshooting. This condition and any error codes should be reported to your technical support representative.
✱	**CPU OK**, **OUTPUTS ENABLED, RUN**	Blinking in unison	CPU is in boot mode and is waiting for a firmware update through a serial port.
✱	**OK**	Blinking Other LEDs off.	CPU in Stop/Halt state; possible watchdog timer fault. Refer to the fault tables. If the programmer cannot connect, cycle power with battery attached and refer to fault tables.
○	**RUN**	Off	CPU is in Stop mode.
●	**OUTPUTS ENABLED**	On	Output scan is enabled.
○	**OUTPUTS ENABLED**	Off	Output scan is disabled.
●	**I/O FORCE**	On	Override is active on a bit reference.
✱	**BATTERY**	Blinking	The Battery Low indication is not supported and should be ignored.
●	**BATTERY**	On	The Battery Failed indication is not supported on the CPU310 and should be ignored. For more information, refer to *Important Product Information,* GFK-2329. ***Note:*** To provide reliable backup, routine maintenance should include scheduled battery replacement. See “Battery Life Estimates” on page 2-13.
●	**SYSTEM FAULT**	On	CPU is in Stop/Faulted mode because a fatal fault has occurred.
✱	**COM1** **COM2**	Blinking Blinking	Signal activity on port.

*After initialization sequence is complete.

Specifications – CPU310

For environmental specifications, see Appendix A of the *PACSystems RX3i System Manual,* GFK-2314.

Battery: Memory retention	For estimated battery life under various conditions, see "Battery Life Estimates," below.
Program storage	Up to 10 Mbytes of battery-backed RAM 10Mbyte of non-volatile flash user memory
Power requirements	+3.3 VDC: 1.25 Amps nominal +5 VDC: 1.0 Amps nominal
Operating Temperature	0 to 60°C (32°F to 140°F)
Floating point	Yes
Boolean execution speed, typical	0.195ms per 1000 Boolean contacts/coils
Time of Day Clock accuracy	Maximum drift of ±2 seconds per day. Can be synchronized to an Ethernet time master within ±2ms of the SNTP time stamp.
Elapsed Time Clock (internal timing) accuracy	±0.01% maximum
Embedded communications	RS-232, RS-485
Serial Protocols supported	Modbus RTU Slave, SNP, Serial I/O
Backplane	Dual backplane bus support: RX3i PCI and 90-30-style serial
PCI compatibility	System designed to be electrically compliant with PCI 2.2 standard
Program blocks	Up to 512 program blocks. Maximum size for a block is 128KB.
Flash memory endurance rating	100,000 write/erase cycles minimum
Memory (For a detailed listing of memory areas, refer to chapter 7.)	*%I and %Q:* 32Kbits for discrete *%AI and %AQ:* configurable up to 32Kwords *%W:* configurable up to the maximum available user RAM Managed memory (*Symbolic and I/O variables combined):* configurable up to 10 Mbytes

Battery Life Estimates for CPU310

To avoid loss of RAM memory contents, routine maintenance procedures should include scheduled replacement of the CPU's lithium battery pack. The following table lists estimates of battery life that can be used to develop a battery replacement schedule.

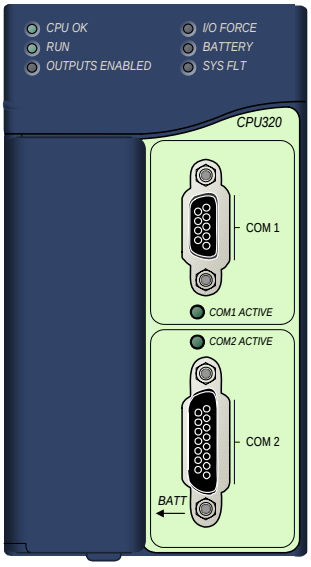
Nominal IC698ACC701 Battery Pack Installed Life

Controller	***Average Temperature***	***Nominal Life with Applied Power On:***	
		100% of the Time	***0% of the Time***
IC695CPU310	20°C (68°F)	5 years	40 days

The IC698ACC701 battery pack has a nominal shelf life of 5 years when stored at an average temperature of 20°C (68°F).

Note: An external Auxiliary Battery Module, IC693ACC302, can be used to provide long-term battery backup for any PACSystems CPU. For details, refer to the datasheet for the Auxiliary Battery Module, GFK-2124.

IC695CPU320



- IC695CPU320: 1 GHz CPU microprocessor

Serial Ports

The CPU has two independent, on-board serial ports, accessed by connectors on the front of the module. Ports 1 and 2 provide serial interfaces to external devices. Either port can be used for firmware upgrades. For serial port pin assignments and details on serial communications, refer to chapter 12.

Indicators

The eight CPU LEDs indicate the operating status of various CPU functions.

LED State ● On ✿ Blinking ○ Off			***CPU Operating State***
●	**CPU OK**	On	CPU has passed its powerup diagnostics and is functioning properly.*
○	**CPU OK**	Off	CPU problem. RUN and OUTPUTS ENABLED LEDs may be blinking in an error code pattern, which can be used by technical support for troubleshooting. This condition and any error codes should be reported to your technical support representative.
✿	**CPU OK**, **OUTPUTS ENABLED, RUN**	Blinking in unison	CPU is in boot mode and is waiting for a firmware update through a serial port.
✿	**OK**	Blinking Other LEDs off.	CPU in Stop/Halt state; possible watchdog timer fault. Refer to the fault tables. If the programmer cannot connect, cycle power with battery attached and refer to fault tables.
○	**RUN**	Off	CPU is in Stop mode.
●	**OUTPUTS ENABLED**	On	Output scan is enabled.
○	**OUTPUTS ENABLED**	Off	Output scan is disabled.
●	**I/O FORCE**	On	Override is active on a bit reference.
●	**BATTERY**	On	Battery has failed or is not attached. ***Note:*** To provide reliable backup, routine maintenance should include scheduled battery replacement. See "Specifications" on page 2-15.
●	**SYSTEM FAULT**	On	CPU is in Stop/Faulted mode because a fatal fault has occurred.
✿	**COM1** **COM2**	Blinking Blinking	Signal activity on port.

*After initialization sequence is complete.

Specifications – CPU320

For environmental specifications, see Appendix A of the *PACSystems RX3i System Manual,* GFK-2314.

Battery: Memory retention	Estimated 30 days using an IC693ACC302 Auxiliary Battery Module at 20ºC. For details on the operation of the Auxiliary Battery Module, refer to the datasheet, GFK-2124. **Note:** The IC698ACC701 Lithium Battery Pack is ***not compatible*** with the CPU320.
Program storage	Up to 64 Mbytes of battery-backed RAM 64 Mbyte of non-volatile flash user memory
Power requirements	+3.3 VDC: 1.0 Amps nominal +5 VDC: 1.2 Amps nominal
Operating Temperature	0 to 60°C (32°F to 140°F)
Floating point	Yes
Boolean execution speed, typical	0.047 ms per 1000 Boolean contacts/coils
Time of Day Clock accuracy	Maximum drift of ±2 seconds per day. Can be synchronized to an Ethernet time master within ±2ms of the SNTP time stamp.
Elapsed Time Clock (internal timing) accuracy	±0.01% maximum
Embedded communications	RS-232, RS-485
Serial Protocols supported	Modbus RTU Slave, SNP, Serial I/O
Backplane	Dual backplane bus support: RX3i PCI and 90-30-style serial
PCI compatibility	System designed to be electrically compliant with PCI 2.2 standard
Program blocks	Up to 512 program blocks. Maximum size for a block is 128KB.
Flash memory endurance rating	100,000 write/erase cycles minimum
Memory (For a detailed listing of memory areas, refer to chapter 7.)	*%I and %Q:* 32Kbits for discrete *%AI and %AQ:* configurable up to 32Kwords *%W:* configurable up to the maximum available user RAM Managed memory (*Symbolic and I/O variables combined):* configurable up to 64 Mbytes

Chapter 3

CPU Configuration

The PACSystems CPU and I/O system is configured using Machine Edition Logic Developer-PLC programming software.

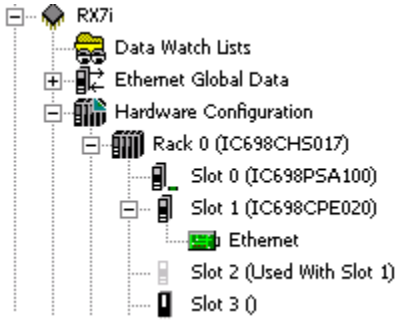
The CPU verifies the physical module and rack configuration at power-up and periodically during operation. The physical configuration must be the same as the programmed configuration. Differences are reported to the CPU alarm processor for configured fault response. Refer to the *Machine Edition Logic Developer-PLC Getting Started Manual*, GFK-1918 and the online help for a description of configuration functions.

Note: A CPE020, CPE030 or CPE040 can be converted to the corresponding redundancy CPU (CRE020, CRE030 or CRE040) by installing different firmware and moving a jumper. Detailed instructions are included in the firmware upgrade kit for the redundancy CPU.

Configuring the CPU

To configure the CPU using the Logic Developer-PLC programming software, do the following:

1. In the Project tab of the Navigator, expand your PACSystems Target, the hardware configuration, and the main rack (Rack 0).
2. Right click the CPU slot and choose Configure. The Parameter Editor window displays the CPU parameters.



Note: An RX7i CPU must be installed in slot 1. The RX3i CPU occupies two slots and can be installed in any pair of slots except the two highest numbered slots in the rack. For details on migrating Series 90 applications to PACSystems, refer to Appendix C.

3. To edit a parameter value, click the desired tab, then click in the appropriate Values field. Refer to “Configuration Parameters” on page 3-2 for information on these fields.
4. Store the configuration to the PLC so these settings can take effect. For details, see “Storing (Downloading) a Configuration” on page 3-17.

Note: The embedded Ethernet Interface (RX7i only) is displayed in a subslot of the CPU slot. For details on configuring the embedded Ethernet Interface, refer to chapter 4.

Configuration Parameters

Settings Parameters

These parameters specify basic operating characteristics of the CPU. For details on how these parameters affect CPU operation, refer to chapter 5.

Settings Parameters	
Passwords	Specifies whether passwords are Enabled or Disabled. Default: Enabled. **Note:** When passwords are disabled, they cannot be re-enabled without clearing PLC memory.
Stop-Mode I/O Scanning	Specifies whether the I/O is scanned while the PLC is in Stop mode. Default: Disabled. (Always Disabled for Redundancy CPU.) **Note:** This parameter corresponds to the I/O ScanStop parameter on a Series 90-70 PLC.
Watchdog Timer (ms)	(ms in 10 ms increments.) Requires a value that is greater than the program sweep time. The watchdog timer is designed to detect "failure to complete sweep" conditions. The CPU restarts the watchdog timer at the beginning of each sweep. The watchdog timer accumulates time during the sweep. The software watchdog timer is useful in detecting abnormal operation of the application program, which could prevent the PLC sweep from completing within the watchdog time period. Valid range: 10 through 2550, in increments of 10. Default: 200. **Note:** For details on setting the watchdog timer in a CPU redundancy system, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308.
Logic/Configuration Power-up Source	Specifies the location/source of the logic and configuration data that is to be used (or loaded/copied into RAM) after each power up. Choices: Always RAM, Always Flash, Conditional Flash. Default: Always RAM.
Data Power-up Source	Specifies the location/source of the reference data that is to be used (or loaded/copied into RAM) after each power up. Choices: Always RAM, Always Flash, Conditional Flash. Default: Always RAM.
Run/Stop Switch	Enables or disables the Run/Stop Mode Switch. Choices: **Enabled:** Enables you to use the physical switch on the PLC to switch the PLC into Stop mode or from Stop mode into Run mode and clear non-fatal faults. **Disabled:** Disables the physical Run/Stop switch on the PLC. Default: Enabled. **Note:** If both serial ports are configured for any protocol other than RTU Slave or SNP Slave, the Run/Stop switch should not be disabled without first must making sure that there is a way to stop the CPU, or take control of the CPU through another device such as the Ethernet module. If the CPU can be set to Stop mode, it will switch the protocol from Serial I/O to the Stop Mode protocol (default is RTU Slave). For details on Stop mode settings, refer to “Port 1 and Port 2 Parameters” on page 3-11.

Settings Parameters	
Memory Protection Switch	Enables or disables the Memory Protect feature associated with the Run/Stop Mode Switch. Choices: **Enabled:** Memory Protect is enabled, which prevents writing to program memory and configuration and forcing or overriding discrete data. **Disabled:** Memory Protect is disabled. Default: Disabled.
Power-up Mode	Selects the CPU mode to be in effect immediately after power-up. Choices: Last, Stop, Run. Default: Last (the mode it was in when it last powered down).
Modbus Address Space Mapping Type	Specifies the type of memory mapping to be used for data transfer between Modbus TCP/IP clients and the PACSystems controller. Choices: **Disabled:** The "Disabled" setting is intended for use in systems containing older Ethernet firmware that does not support Modbus TCP. **Standard Modbus Addressing:** Causes the Ethernet firmware to use the standard map, which is displayed on the Modbus TCP Address Map tab. Default: Disabled For details on the PACSystems implementation of Modbus/TCP server, refer to *TCP/IP Communications for PACSystems,* GFK-2224.

Modbus TCP Address Map

This read-only tab displays the standard mapping assignments between Modbus address space and the CPU address space. All Ethernet modules and daughterboards in the PACSystems controller use Modbus-to-PLC address mapping based on this map.

Modbus Register	The Modbus protocol uses five reference table designations: 0xxxx Coil Table. Mapped to the %Q table in the CPU. 1xxxx Input Discrete Table. Mapped to the %I table in the CPU. 3xxxx Input Register Table. Mapped to the %AI register table in the CPU. 4xxxx Holding Register Table. Mapped to the %R table in the CPU. 6xxxx File Access Table. Mapped to the %W table in the CPU.
Start Address	Lists the beginning address of the mapped region.
End Address	Lists the ending address of the mapped region. For word memory types (%AI, %R and %W) the highest address available is configured on the Memory tab.
PLC Memory	Lists the memory type of the mapped region.
Length	Displays the length of the mapped region.

Scan Parameters

These parameters determine the characteristics of CPU sweep execution.

Scan Parameters	
Sweep Mode	The sweep mode determines the priority of tasks the CPU performs during the sweep and defines how much time is allotted to each task. The parameters that can be modified vary depending on the selection for sweep mode. The Controller Communications Window, Backplane Communications Window, and Background Window phases of the PLC sweep can be run in various modes, based on the PLC sweep mode. Choices: ■ Normal mode: The PLC sweep executes as quickly as possible. The overall PLC sweep time depends on the logic program and the requests being processed in the windows and is equal to the time required to execute the logic in the program plus the respective window timer values. The window terminates when it has no more tasks to complete. This is the default value. ■ Constant Window mode: Each window operates in a Run-to-Completion mode. The PLC alternates among three windows for a time equal to the value set for the window timer parameter. The overall PLC sweep time is equal to the time required to execute the logic program plus the value of the window timer. This time may vary due to sweep-to-sweep differences in the execution of the program logic. ■ Constant Sweep mode: The overall PLC sweep time is fixed. Some or all of the windows at the end of the sweep might not be executed. The windows terminate when the overall PLC sweep time has reached the value specified for the Sweep Timer parameter.
Logic Checksum Words	The number of user logic words to use as input to the checksum algorithm each sweep. Valid range: 0 through 32760, in increments of 8. Default: 16.
Controller Communication Window Mode	(Available only when Sweep Mode is set to *Normal.)* Execution settings for the Controller Communications Window. Choices: ■ Complete: The window runs to completion. There is no time limit. ■ Limited: Time sliced. The maximum execution time for the Controller Communications Window per scan is specified in the Controller Communications Window Timer parameter. Default: Limited. **Note:** This parameter corresponds to the Programmer Window Mode parameter on a Series 90-70 PLC.
Controller Communications Window Timer (ms)	(Available only when Sweep Mode is set to *Normal.* Read-only if the Controller Communications Window Mode is set to Complete.) The maximum execution time for the Controller Communications Window per scan. This value cannot be greater than the value for the watchdog timer. The valid range and default value depend on the Controller Communications Window Mode: ■ Complete: There is no time limit. ■ Limited: Valid range: 0 through 255 ms. Default: 10. **Note:** This parameter corresponds to the Programmer Window Timer parameter on a Series 90-70 PLC.

Scan Parameters	
Backplane Communication Window Mode	(Available only when Sweep Mode is set to *Normal.*) Execution settings for the Backplane Communications Window. Choices: Complete: The window runs to completion. There is no time limit. Limited: Time sliced. The maximum execution time for the Backplane Communications Window per scan is specified in the Backplane Communications Window Timer parameter. Default: Complete.
Backplane Communications Window Timer (ms)	(Available only when Sweep Mode is set to *Normal.* Read-only if the Backplane Communications Window Mode is set to *Complete.*) The maximum execution time for the Backplane Communications Window per scan. This value can be greater than the value for the watchdog timer. The valid range and the default depend on the Backplane Communications Window Mode: ■ Complete: There is no time limit. The Backplane Communications Window Timer parameter is read-only. ■ Limited: Valid range: 0 through 255 ms. Default: 255. (10ms for Redundancy CPUs.)
Background Window Timer (ms)	(Available only when Sweep Mode is set to *Normal.*) The maximum execution time for the Background Communications Window per scan. This value cannot be greater than the value for the watchdog timer. Valid range: 0 through 255 Default: 0 (5ms for Redundancy CPUs)
Sweep Timer (ms)	(Available only when Sweep Mode is set to *Constant Sweep.*) The maximum overall PLC scan time. This value cannot be greater than the value for the watchdog timer. Some or all of the windows at the end of the sweep might not be executed. The windows terminate when the overall PLC sweep time has reached the value specified for the Sweep Timer parameter. Valid range: 5 through 2550, in increments of 5. If the value typed is not a multiple of 5ms, it is rounded to the next highest valid value. Default: 100.
Window Timer (ms)	(Available only when Sweep Mode is set to *Constant Window.*) The maximum combined execution time per scan for the Controller Communications Window, Backplane Communications Window, and Background Communications Window. This value cannot be greater than the value for the watchdog timer. Valid range: 3 through 255, in increments of 1. Default: 10.
Number of Last Scans	(Available only for CPUs with firmware version 1.5 and greater.) The number of scans to execute after the PACSystems CPU receives an indication that a transition from Run to Stop mode should occur. (Used for Stop and Stop Fault, but not Stop Halt.) Choices: 0, 1, 2, 3, 4, 5. Default: 0 when creating a new PACSystems target. 0 when converting a Series 90-70 target to a PACSystems target. 1 when converting a Series 90-30 target to a PACSystems target.

Memory Parameters

The PACSystems user memory contains the application program, hardware configuration (HWC), registers (%R), bulk memory (%W), analog inputs (%AI), analog outputs (%AQ), and managed memory.

Managed memory consists of allocations for symbolic variables and I/O variables. The symbolic variables feature allows you to create variables without having to manually locate them in memory. An I/O variable is a symbolic variable that is mapped to a module's inputs and outputs in the hardware configuration. For details on using symbolic variables and I/O variables, refer to chapter 7.

The amount of memory allocated to the application program and hardware configuration is automatically determined by the actual program (including logic C data, and %L and %P), hardware configuration (including EGD and AUP), and symbolic variables created in the programming software. The rest of the user memory can be configured to suit the application. For example, an application may have a relatively large program that uses only a small amount of register and analog memory. Similarly, there might be a small logic program but a larger amount of memory needed for registers and analog inputs and outputs.

Appendix B provides a summary of items that count against user memory.

Calculation of Memory Required for Managed Memory

The total number of bytes required for symbolic and I/O variables is calculated as follows:

[((number of symbolic discrete bits) * 3) / (8 bits/byte)]

+ [((number of I/O discrete bits) * Md) / (8 bits/byte)]

+ [(number of symbolic words) * (2 bytes/word)]

+ [(number of I/O words) * (Mw bytes/word)]

Md = 3 or 4. The number of bits is multiplied by 3 to keep track of the force, transition, and value of each bit. If point faults are enabled, the number of I/O discrete bits is multiplied by 4.

Mw = 2 or 3. There are two 8-bit bytes per 16-bit word. If point faults are enabled, the number of bytes is multiplied by 3 because each I/O word requires an extra byte.

Calculation of Total User Memory Configured

The total amount of configurable user memory (in bytes) configured in the CPU is calculated as follows:

total managed memory (bytes)

+ total reference words * (2 bytes/word)

+ *[if Point Faults are enabled]* (total words of %AI memory + total words of %AQ memory) * (1 byte / word)

+ *[if Point Faults are enabled]* (total bits of %I memory + total bits of %Q memory) / 8 bits/byte)

Note: The total reference points is considered system memory and is not counted against user memory.

Memory Allocation Configuration

Memory Parameters	
Reference Points	
%I Discrete Input, %Q Discrete Output, %M Internal Discrete, %S System, %SA System, %SB System, %SC System, %T Temporary Status, %G Genius Global	The upper range for each of these memory types. Read only.
Total Reference Points	Read only. Calculated by the programming software.
Reference Words	
%AI Analog Input	Valid range: 0 through 32,640 words. Default: 64
%AQ Analog Output	Valid range: 0 through 32,640 words. Default: 64
%R Register Memory	Valid range: 0 through 32,640 words. Default: 1024.
%W Bulk Memory	Valid range: 0 through maximum available user RAM. Increments of 2048 words. Default: 0.
Total Reference Words	Read only. Calculated by the programming software.
Managed Memory	
Symbolic Discrete (Bits)	The configured number of bits reserved for symbolic discrete variables. Valid range: 0 through 83,886,080 in increments of 32768 bits. Default: 32,768.
Symbolic Non-Discrete (Words)	The configured number of 16-bit register memory locations reserved for symbolic non-discrete variables. Valid range: 0 through 5,242,880 in increments of 2048 words. Default: 65,536.
I/O Discrete (Bits)	The configured number of bits reserved for discrete IO variables. Valid range: 0 through 83,886,080 in increments of 32768 bits. Default: 0
I/O Non-Discrete (Words)	The configured number of 16-bit register memory locations reserved for non-discrete IO variables. Valid range: 0 through 5,242,880 in increments of 2048 words. Default: 0
Total Managed Memory Required (Bytes)	Read only. See page 3-6 for calculation.
Total User Memory Required (Bytes)	Read only. See page 3-6 for calculation.
Point Fault References	The Point Fault References parameter must be enabled if you want to use fault contacts in your logic. Assigning point fault references causes the CPU to reserve additional memory. When you download both the HWC and the logic to the PLC, the download routine checks if there are fault contacts in the logic and if there are, it checks if the HWC to download has the Point Fault References parameter set to Enabled. If the parameter is Disabled, an error is displayed in the Feedback Zone. When you download only logic to the PLC, the download routine checks if there are fault contacts in the logic and if there are, it checks if the HWC on the PLC has the Point Fault References parameter set to Enabled. If the parameter is Disabled, an error is displayed in the Feedback Zone.

Fault Parameters

You can configure each fault action to be either diagnostic or fatal.

A ***diagnostic fault*** does not stop the PLC from executing logic. It sets a diagnostic variable and is logged in a fault table.

A ***fatal fault*** transitions the PLC to the Stop Faulted mode. It also sets a diagnostic variable and is logged in a fault table.

Fault Parameters	
Loss of or Missing Rack	(Fault group 1.) When BRM failure or loss of power loses a rack or when a configured rack is missing, system variable #LOS_RCK (%SA12) turns ON. (To turn it OFF, fix the hardware problem and cycle power on the rack.) Default: Diagnostic.
Loss of or Missing I/O Controller	(Fault group 2.) When a Bus Controller stops communicating with the PLC or when a configured Bus Controller is missing, system variable #LOS_IOC (%SA13) turns ON. (To turn it OFF, replace the module and cycle power on the rack containing the module.) Default: Diagnostic.
Loss of or Missing I/O Module	(Fault group 3.) When an I/O module stops communicating with the PLC CPU or a configured module is missing, system variable #LOS_IOM (%SA14) turns ON. (To turn it OFF, replace the module and cycle power on the rack containing the module.) Default: Diagnostic.
Loss of or Missing Option Module	(Fault group 4.) When an option module stops communicating with the PLC CPU or a configured option module is missing, system variable #LOS_SIO (%SA15) turns ON. (To turn it OFF, replace the module and cycle power on the rack containing the module.) Default: Diagnostic.
System Bus Error	(Fault group 12.) When a bus error occurs on the backplane, system variable #SBUS_ER (%SA32) turns ON. (To turn it OFF, cycle power on the main rack.) Default: Fatal.
I/O Controller or I/O Bus Fault	(Fault group 9.) When a Bus Controller reports a bus fault, a global memory fault, or an IOC hardware fault, system variable #IOC_FLT (%SA22) turns ON. (To turn it OFF, cycle power on the rack containing the module when the configuration matches the hardware after a download.) Default: Diagnostic.
System Configuration Mismatch	(Fault group 11.) When a configuration mismatch is detected during system power-up or during a download of the configuration, system variable #CFG_MM (%SA9) turns ON. (To turn it OFF, power up the PLC when no mismatches are present or download a configuration that matches the hardware.) This parameter determines the fault action when the CPU is ***not running.*** If a system configuration mismatch occurs when the CPU is in Run mode, the fault action will be Diagnostic. This prevents the running CPU from going to STOP/FAULT mode. To override this behavior, see “Configuring the CPU to Stop Upon Loss of a Critical Module” on page 3-9. Default: Fatal.
Recoverable Local Memory Error	***Redundancy CPUs only.*** (Fault group 38) Determines whether a single-bit ECC error causes the CPU to stop or allows it to continue running. Choices: Diagnostic, Fatal. Default: Diagnostic. **Note:** When a multiple-bit ECC error occurs, a Fatal Local Memory Error fault (error code 169) is logged in the CPU Hardware Fault Group (group number 13).

Fault Parameters	
CPU Over Temperature	(Fault group 24, error code 1.) When the operating temperature of the CPU exceeds the normal operating temperature, system variable #OVR_TMP (%SA8) turns ON. (To turn it OFF, clear the controller fault table or reset the PLC.) Default: Diagnostic.
Controller Fault Table Size	(Read-only.) The maximum number of entries in the Controller Fault Table. Value set to 64.
I/O Fault Table Size	(Read-only.) The maximum number of entries in the I/O Fault Table. Value set to 64.

Configuring the CPU to Stop Upon the Loss of a Critical Module

In some cases, you may want to override the Run mode behavior of the System Configuration Mismatch fault. A given module may be critical to the PLC's ability to properly control a process. In this case, if the module fails then it may be better to have the CPU go to stop mode, especially if the CPU is acting as a backup unit in a redundant system.

One way to cause the CPU to stop is to set the configured action for a Loss-of-Module fault to *Fatal* so that the CPU stops if a module failure causes a loss-of-module fault. The correct loss-of-module fault must be chosen for the critical module of interest: I/O controller, I/O module, and Option module. The Ethernet communications module is an example of an Option module.

This approach has a couple of disadvantages. First, it applies to all modules of that category, which may include modules that are not critical to the process. Second, it relies on the content of the fault table. If the table is cleared via program logic or user action, the CPU will not stop.

In systems that use Ethernet Network Interface Units (ENIU) for remote I/O, a critical module of interest may be the Ethernet module that provides the network connection to the ENIU. Other techniques can be used to provide a more selective response to an Ethernet module failure than the Loss-of-Option module fault. One technique is to use application logic to monitor the Ethernet Interface Status bits, which are described in "Monitoring the Ethernet Interface Status Bits" in the *TCP/IP Ethernet for PACSystems User's Manual,* GFK-2224. If the logic determined that a critical Ethernet module was malfunctioning, it could execute SVC_REQ #13 to stop the CPU.

Since the ENIU uses Ethernet Global Data to communicate with the PACSystems CPU, another selective technique is to monitor the Exchange Status Words to determine the health of individual EGD exchanges. For details on this status word, refer to "Exchange Status Word Error Codes" in GFK-2224. Because the types of errors indicated by the exchange status word may be temporary in nature, stopping the CPU may not be an appropriate response for these errors. Nevertheless, the status could be used to tailor the application's response to changing conditions in the EGD network.

In some cases the critical module may reside in an expansion rack. In that case, in addition to the loss-of-module fault, it is recommended to set the Loss-of-Rack fault to Fatal. Then if the rack fails or loses power, the CPU will go to stop mode.

Redundancy Parameters (Redundancy CPUs Only)

These parameters apply only to redundancy CPUs such as IC698CRE020. For details on configuring CPU for redundancy, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308.

Transfer List

These parameters apply only to redundancy CPUs such as IC698CRE020. For details on configuring CPU for redundancy, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308.

Port 1 and Port 2 Parameters

These parameters configure the operating characteristics of the CPU serial ports. Ports 1 and 2 have the same set of configuration parameters. The protocol (Port Mode) that is selected determines the parameters that can be set for each port.

<table>
<tr><th colspan="2">Port Parameters</th></tr>
<tr><td>Port Mode</td><td>
The protocol to execute on the serial port. Determines the list of parameters displayed on the Port tab. Only the parameters required by the selected protocol are displayed.
Choices:
■ RTU Slave mode: Reserved for the use of the Modbus RTU Slave protocol. This mode also permits connection to the port by an SNP master, such as the Winloader utility or the programming software.
■ Message mode: The port is open for user logic access. This mode enables C language blocks to perform serial port i/o operations via the C Runtime Library functions.
■ Available: The port is not to be used by the PLC firmware.
■ SNP Slave: Reserved for the exclusive use of the SNP slave. This mode permits connection to the port by an SNP master, such as the Winloader utility or the programming software.
■ Serial I/O: Enables you to perform general-purpose serial communications by using COMMREQ functions.
Default: RTU Slave.
To select the Stop mode protocol, set the Specify Stop Mode parameter to Yes, and then select a protocol for Stop mode. If the Stop mode protocol is different from the Port mode protocol, you can set parameters for the Stop mode protocol.
If you do not select a Stop mode protocol, the default protocol with default parameter settings is used.
<table>
<tr><th>Port (Run) Mode</th><th>Stop Mode</th></tr>
<tr><td>RTU Slave</td><td>Choices: SNP Slave, RTU Slave
Default: RTU Slave.</td></tr>
<tr><td>Message Mode</td><td>Choices: SNP Slave, RTU Slave
Default: RTU Slave.</td></tr>
<tr><td>Available</td><td>Available</td></tr>
<tr><td>SNP Slave</td><td>SNP Slave</td></tr>
<tr><td>Serial I/O</td><td>Choices: SNP Slave, RTU Slave
Default: RTU Slave.</td></tr>
</table>
Note: If both serial ports are configured for any protocol other than RTU Slave or SNP Slave, the Run/Stop switch should not be disabled without first making sure that there is a way to stop the CPU, or take control of the CPU through another device such as the Ethernet module. The Serial I/O protocol is only active when the CPU is in run mode. If the CPU can be set to Stop mode, it will switch the protocol from Serial I/O to the Stop Mode (default is RTU Slave). If an SNP Master, such as the programming software in Serial mode, begins communicating on a port, the RTU protocol automatically switches to SNP Slave. As long as the CPU can be stopped, the port's protocol can be auto-switched to one that enables serial programmer connection.
If the Ethernet module is available, you can control the CPU by connecting the Machine Edition programming software to the Ethernet port.
</td></tr>
</table>

Port Parameters	
Station Address	(RTU Slave only) ID for the RTU Slave. Valid range: 1 through 247. Default: 1. **Note:** You should avoid using station address 1 for any other Modbus slave in a PACSystems control system because the default station address for the CPU is 1. The CPU uses the default address in two situations: 1. If you power up without a configuration, the default station address of 1 is used. 2. When the Port Mode parameter is set to Message Mode, and Modbus becomes the protocol in stop mode, the station address defaults to 1. In either of these situations, if you have a slave configured with a station address of 1, confusion may result when the CPU responds to requests intended for that slave. **Note:** The least significant bit of the first byte must be 0. For example, in a station address of 090019010001, 9 is the first byte.
Data Rate	(All Port Modes except Available.) Data rate (bits per second) for the port. Choices: 1200 Baud, 2400 Baud, 4800 Baud, 9600 Baud, 19.2k Baud, 38.4k Baud, 57.6k Baud, 115.2k Baud. Default: 19.2k Baud.
Data Bits	(Available only when Port Mode is set to Message mode or Serial I/O.) The number of bits in a word for serial communication. SNP uses 8-bit words. Choices: 7, 8. Default: 8.
Flow Control	(RTU slave, Message Mode, or Serial I/O.) Type of flow control to be used on the port. Choices: ▪ For Serial I/O Port Mode: None, Hardware, Software (XON/XOFF). ▪ For all other Port Modes: None, Hardware. Default: None. **Note:** The Hardware flow-control is RTS/CTS crossed.
Parity	(All Port Modes except Available.) The parity used in serial communication. Can be changed if required for communication over modems or with a different SNP master device. Choices: None, Odd, Even. Default: Odd.
Stop bits	(Available only when Port Mode is set to Message Mode, SNP Slave or Serial I/O.) The number of stop bits for serial communication. SNP uses 1 stop bit. Choices: 1, 2. Default: 1.
Physical Interface	(All port modes except Available.) The type of physical interface that this protocol is communicating over. Choices: ■ 2-wire: There is only a single path for receive and transmit communications. The receiver is disabled while transmitting. ■ 4-wire: There is a separate path for receive and transmit communications and the transmit line is driven only while transmitting. ■ 4-wire Transmitter on: There is a separate path for receive and transmit communications and the transmit line is driven continuously. Note that this choice is not appropriate for SNP multi-drop communications, since only one device on the multi-drop line can be transmitting at a given time. Default: 4-wire Transmitter On.

Port Parameters	
Turn Around Delay Time (ms)	(Available only when Port Mode is set to SNP Slave.) The Turn Around Delay Time is the minimum time interval required between the reception of a message and the next transmission. In 2-wire mode, this interval is required for switching the direction of data transmission on the communication line. Valid range: 0 through 2550 ms, in increments of 10. Default: 0.
Timeout (s)	(Available only when Port Mode is set to SNP Slave.) The maximum time that the slave will wait to receive a message from the master. If a message is not received within this timeout interval, the slave will assume that communications have been disrupted, and then it will wait for a new attach message from the master. Valid range: 0 through 60 seconds. Default: 10.
SNP ID	(Available only when Port Mode is set to SNP Slave.) The port ID to be used for SNP communications. In SNP multi-drop communications, this ID is used to identify the intended receiver of a message. This parameter can be left blank if communication is point to point. To change the SNP ID, click the values field and enter the new ID. The SNP ID is up to seven characters long and can contain the alphanumeric characters (A through Z, 0 through 9) or the underline (_).
Specify Stop Mode	(All port modes except Available.) Determines whether you accept the default stop mode or set it yourself. Choices: **No:** The default stop mode is used. **Yes:** The stop mode parameters appear and you can select the stop mode. If you set the stop mode to the same protocol as the run mode, then the other stop mode parameters are read-only and are set to the same values as for the run mode. Default: No.
Stop Mode	(Available only when Specify stop mode is set to Yes.) The stop mode protocol to execute on the serial port. If you set the stop mode to the same protocol as for the run mode, then the other stop mode parameters are read-only and are set to the same values as for the run mode. Choices: ■ SNP Slave: Reserved for the exclusive use of the SNP slave. This is the only valid choice if the Port Mode is set to SNP Slave. ■ RTU Slave: Reserved for the exclusive use of the Modbus RTU Slave protocol. Default: ■ When the Port Mode is set to SNP Slave: SNP Slave (only valid choice). ■ When the Port Mode is set to RTU Slave, Message Mode, or Serial I/O: RTU Slave.
Turn Around Delay Time (ms)	(Available only when Stop Mode is set to SNP Slave.) The Turn Around Delay Time is the minimum time interval required between the reception of a message and the next transmission. In 2-wire mode, this interval is required for switching the direction of data transmission on the communication line. Valid range: 0 through 2550 ms, in increments of 10. Default: ■ When the Stop Mode is different from the Port Mode: 0 ms. ■ When the Stop Mode is the same as the Port Mode: the value is read-only and is set to the same value as the Turn Around Delay Time for the Port Mode.

Port Parameters	
Timeout (s)	(Available only when Stop Mode is set to SNP Slave.) The maximum time that the slave will wait to receive a message from the master. If a message is not received within this timeout interval, the slave will assume that communications have been disrupted, and then it will wait for a new attach message from the master. Valid range: 0 through 60 seconds. Default: ■ When the Stop Mode is different from the Port Mode: 10 seconds. ■ When the Stop Mode is the same as the Port Mode: the value is read-only and is set to the same value as the Timeout for the Port Mode.
SNP ID	(Available only when Stop Mode is set to SNP Slave.) The port ID to be used for SNP communications. In SNP multi-drop communications, this ID is used to identify the intended receiver of a message. This parameter can be left blank if communication is point to point. To change the SNP ID, click the values field and enter the new ID. The SNP ID is up to seven characters long and can contain the alphanumeric characters (A through Z, 0 through 9) or the underline (_). Default: ■ When the Stop Mode is different from the Port Mode: the default is blank. ■ When the Stop Mode is the same as the Port Mode: the value is read-only and is set to the same value as the SNP ID for the Port Mode.
Station Address	(Available only when Stop Mode is set to RTU slave.) ID for the RTU Slave. Valid range: 1 through 247. Default: ■ When the Stop Mode is different from the Port Mode: 1. ■ When the Stop Mode is the same as the Port Mode: the value is read-only and is set to the same value as the Station Address for the Port Mode.

Scan Sets Parameters

You can create multiple sets of asynchronous I/O scans, with a unique scan rate assigned to each scan set. You can assign up to 31 scan sets for a total of 32. Scan set 1 is the standard scan set where I/O is scanned once per sweep. Each module is assigned to a scan set in the module's configuration. Scan Set 1 is the default scan set.

Scan Set Parameters	
Number	A sequential number from 1 to 32 is automatically assigned to each scan set. Scan set 1 is reserved for the standard scan set.
Scan Type	Determines whether the scan set is enabled (as a fixed scan) or is disabled. Choices: Disabled, Fixed Scan. Default: Disabled.
Number of Sweeps	(Editable only when the Scan Type is set to Fixed Scan.) The scan rate of the scan set. Double-click the field, then select a value. A value of 0 prevents the I/O from being scanned. Valid range: 0 through 64. Default: 1.
Output Delay	(Editable only when the Number of Sweeps is non-zero.) The number of sweeps that the output scan is delayed after the input scan has occurred. Double-click on field, then select a value. Valid range: 0 to (number of Sweeps - 1) Default: 0.
Description	(Editable only when the Scan Type is set to Fixed Scan.) Brief description of the scan set (32 characters maximum).

Power Consumption Parameters

The programming software displays the power consumed by the CPU (in Amps) for each voltage provided by the power supply.

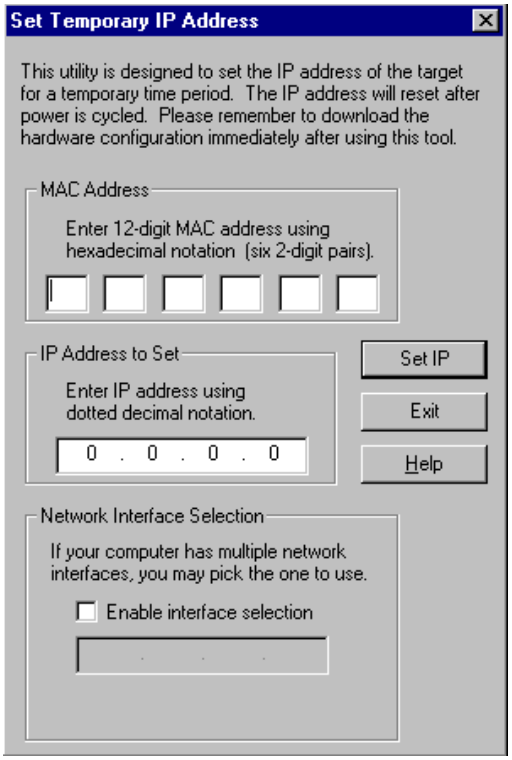
Setting a Temporary IP Address

To initiate Ethernet communications between the programming software and the PACSystems, you first need to set an IP address. You can use the Set Temporary IP Address utility to specify an IP address or download a hardware configuration with an IP address through a serial port.

The following restrictions apply when using the Set Temporary IP Address utility:

- To use the Set Temporary IP Address utility, the PLC CPU must not be in RUN mode. IP address assignment over the network will not be processed until the CPU is stopped and is not scanning outputs.
- The Set Temporary IP Address utility does not function if communications with the networked PACSystems target travel through a router. The Set Temporary IP Address utility can be used if communications with the networked PACSystems target travel across network switches and hubs.
- The current user logged on the computer running the Set Temporary IP Address utility must have full administrator privileges.
- The target PACSystems must be located on the same local sub-network as the computer running the Set Temporary IP Address utility. The sub-network is specified by the computer's subnet mask and the IP addresses of the computer and the PACSystems Ethernet Interface.

Note: To set the IP address, you will need the MAC address of the Ethernet Interface.



1. Connect the PACSystems to the Ethernet network.
2. In the Project tab of the Navigator, right click the PACSystems target, choose Offline Commands, and then choose Set Temporary IP Address. The Set Temporary IP Address dialog box appears.
3. In the Set Temporary IP Address dialog box, do the following:
   - Specify the MAC address.
   - In the IP Address to Set box, specify the temporary IP address you want to set on the PACSystems.
   - If necessary, select the Enable Network Interface Selections check box and specify the IP address of the network interface on which the PACSystems is located.
4. When the fields are properly configured, click the Set IP button.
5. The IP Address of the specified PACSystems will be set to the indicated address. This may take up to a minute.

After the programmer is connected, the actual IP address for the Ethernet interface, which is set in the hardware configuration, should be downloaded to the controller. The temporary IP address remains in effect until the Ethernet interface is restarted, power-cycled or until the hardware configuration is downloaded or cleared.

Cautions

The temporary IP address set by the Set IP utility is not retained through a power cycle. To set a permanent IP Address, you must set the target's IP Address property and download (store) HWC to the PACSystems.

The Set Temporary IP Address utility can assign a temporary IP address even if the target Ethernet Interface has previously been configured to a non-default IP address. (This includes overriding an IP address previously configured by the programmer.)

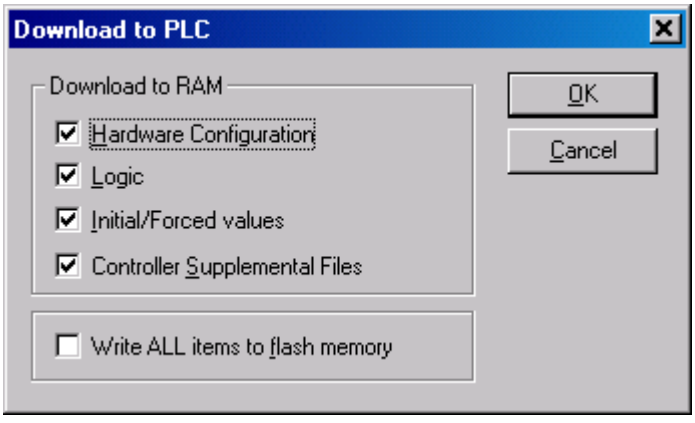
Use this IP Address assignment mechanism with care.

Storing (Downloading) Hardware Configuration

A PACSystems control system is configured by creating a configuration file in the programming software, then transferring (downloading) the file from the programmer to the CPU via serial port1, serial port 2, or an Ethernet Interface. If you use a serial port, it must be configured as RTU Slave (default) or SNP Slave.

The CPU stores the configuration file in its non-volatile RAM memory. After the configuration is stored, I/O scanning is enabled or disabled according to the newly stored configuration parameters.

1. If you are using an Ethernet Interface to store the hardware configuration to the PACSystems, you must set the IP address in the Ethernet Interface using the Initial IP Address utility (see page 3-16).
2. Make sure the CPU is in Stop mode.
3. In Logic Developer-PLC software, go to the Project tab of the Navigator, right click the Target node, and choose Download to PLC.
4. In the Download to PLC dialog box, select the items to download and click OK.



Note: If you download to a PACSystems target that already has a project on it, the existing project is overwritten.

Note: If I/O variables are configured, hardware configuration and logic cannot be stored independently. They must be stored at the same time.

Chapter 4

Embedded Ethernet Interface Configuration

Before you can use the embedded Ethernet Interface, you must configure it using the programming software. The configuration process includes:

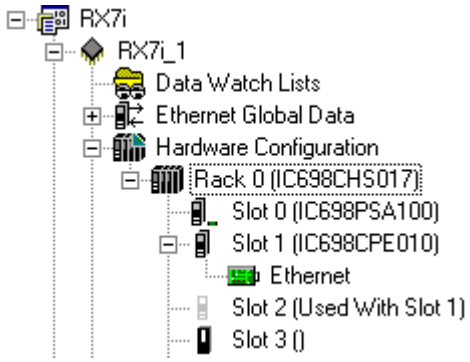
- Assign a temporary IP address for initial network operation, such as connecting the programmer to download the hardware configuration. Temporary IP address assignment is described in *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.
- Configure the characteristics of the Ethernet interface.
- Configure Ethernet Global Data (if used). Ethernet Global Data (EGD) configuration is described in *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.
- Set up the RS-232 port for Local Station Manager operation. This is part of the basic Ethernet Interface configuration (optional).
- Configure Advanced User Parameters (optional). The default Advanced User Parameter values that be changed only in exceptional circumstances by experienced users. The Advanced User Parameters definitions and configuration are described in *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.
- Setting up the PLC for Modbus/TCP Server operation (Optional). See *TCP/IP Ethernet Communications for PACSystems,* GFK-2224 for information about configuring Modbus/TCP Server operation.

Note: Only RX7i CPUs provide an embedded Ethernet interface. For information on configuring the rack-based RX7i and RX3i Ethernet Interface modules, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.

Configuring the Embedded Ethernet Interface

For details on configuring a rack system using Logic Developer-PLC software, refer to the software online Help. For information on CPU parameters, refer to chapter 2.

1. In the Project tab of the Navigator, expand your PACSystems Target, the hardware configuration, and the main rack (Rack 0).
2. Expand the CPU slot (Slot 1). The Ethernet Interface daughterboard is displayed as “Ethernet”.
3. Right click the daughterboard slot and choose Configure. The Parameter Editor window displays the Ethernet Interface parameters.



4. The Settings and RS-232 Port (Station Manager) tabs contain parameters directly related to the embedded Ethernet Interface’s functionality. Some fields in the Settings tab must be filled in. The default settings for the RS-232 tabs can be used for most applications. Refer to “Configuration Parameters” on page 4-3 for information on these fields. We recommend leaving the serial port parameters at default settings.
5. Save this configuration and download it to the CPU so these settings can take effect.

Configuration Parameters

Ethernet Settings

The parameters on this tab are used to configure Ethernet communications.

Settings Tab	
Configuration Mode	This is fixed as TCP/IP.
Adapter Name	This field is set to the rack and slot location of the Ethernet Interface and cannot be changed. For the embedded Ethernet Interface, the rack and slot location is the same as its CPU (0.1.0 for rack 0, slot 1, subslot 0).
Use BOOTP for IP Address	This selection specifies whether the Ethernet must obtain its working IP address over the network via BOOTP. When set to False (= do not use BOOTP), the IP Address value must be configured (see IP Address parameter, below). When set to True, the IP Address parameter is set to 0.0.0.0 and cannot be edited.
IP Address, Subnet Mask, and Gateway IP Address	These values should be assigned by the network administrator in charge of your network. It is important that these parameters are correct, otherwise the Ethernet Interface may be unable to communicate on the network and/or network operation may be corrupted. It is especially important that each node on the network is assigned a *unique* IP address. However, if you have no network administrator and are using a simple *isolated network* with no gateways, you can use the following range of values for the assignment of local IP addresses 10.0.0.1 First PLC 10.0.0.2 Second PLC 10.0.0.3 Third PLC . . . 10.0.0.255 PLC Programmer TCP or host Also, in this case, set the subnet mask, and gateway IP address to 0.0.0.0. **Note** If the isolated network is connected to another network, the IP addresses 10.0.0.1 through 10.0.0.255 must not be used and the subnet mask, and gateway IP address must be assigned by the network administrator. The IP addresses must be assigned so that they are compatible with the connected network. For more information on addressing, refer to “Network Administration Support” in *TCP/IP Ethernet Communications for PACSystems,* GFK-2224. See also “Determining If an IP Address is Already Being Used” on page 4-7.
Name Server IP Address	This parameter must be set to 0.0.0.0.
Max Web Server Connections	The maximum number of web server connections. This value corresponds to the number of TCP connections allocated for use by the web server, rather than the number of web clients. Valid range is 0 through 16. Default is 2. **Note** The combined number of web server and FTP server connections is 16. For example, if you have 2 FTP server connections, you can have a maximum of 14 web server connections.
Max FTP Server Connections	The maximum number of FTP server connections. This value corresponds to the number of TCP connections allocated for use by the FTP server, rather than the number of FTP clients. Each FTP client uses two TCP connections when an FTP connection is established. Valid range is 0 through 16. Default is 2.

Settings Tab	
Network Time Sync	The method used to synchronize the real-time clocks over the network. The choices are None (for no network time synchronization) and SNTP (for synchronization to remote SNTP servers on the network). If None is selected, the time stamp value for a consumed EGD exchange is obtained from the local clock of the producing Controller or PLC. Time stamps of exchanges produced by a PLC with this setting are not synchronized with the time stamps of exchanges produced by other PLCs. For more information, see "Timestamping of Ethernet Global Data Exchanges" in chapter 4 of *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.
Status Address	The reference memory location that receives LAN Interface Status (LIS) bits (16 bits) and the Channel Status bits (64 bits). The Channel Status bits are always located immediately following the LAN Interface Status bits. The Status address can be assigned to %I, %Q, %R, %W, %AI or %AQ memory. The default value is the next available %I address. **Notes** Do not use the 80 bits assigned to the LAN Interface Status (LIS) bits and Channel Status bits for other purposes, or your data will be overwritten. For definitions of the LIS portion of the Ethernet Status data, refer to "Diagnostics" in *TCP/IP Ethernet Communications for PACSystems,* GFK-2224. If the Ethernet Interface's Variable Mode property is set to True, the Status Address parameter is removed from the Settings tab. Instead, Ethernet Status references must be defined as I/O variables on the Terminals tab (see Terminals Tab, below)
Length	The sum of the LIS bits and the Channel Status bits. Automatically set to 80 bits (for %I and %Q Status address locations) or 5 words (for %R, %W %AI, and %AQ Status address locations).
Redundant IP	Enabling this feature allows the Ethernet Interface to share a redundant IP address with the corresponding Ethernet Interface in a redundant system. When this parameter is enabled, a Redundant IP Address must be entered. **Note** When the Ethernet interface switches from Active mode to Backup mode, in addition to deactivating the Redundant IP address, the interface also deactivates all Modbus Server connections that are using the Redundant IP address. As a result, whenever a role switch or control-takeover occurs, Modbus/TCP Client applications will need to reconnect to the Redundant IP address to re-establish communications with the redundancy system. The operation from the client side is the same as losing connection with a normal Modbus/TCP server. Default: Disable
Redundant IP Address	(Available only when the Redundant IP parameter is set to Enable.) An optional IP Address that will be shared with another device on the network in a redundant system. Both devices must use the same subnet mask. For more information about Ethernet redundancy, see *TCP/IP Ethernet Communications for PACSystems*, GFK-2224.). This IP address is assigned in addition to the device's primary IP address. The Redundant IP Address should be assigned by the person responsible for your network. If the IP address is improperly set, your device may not be able to communicate on the network and could disrupt network communications. Valid range: x.x.x.x, where x ranges from 1 through 255. Default: 0.0.0.0. **Note** Each Ethernet Interface has its own unique IP address, which is used to communicate exclusively with it. The redundant IP address must not be the same as the unique IP address of either Ethernet Interface.

Settings Tab	
I/O Scan Set	The scan set to be assigned to the Ethernet daughterboard. Scan sets are defined in the CPU's Scan Sets tab. The valid range is 1 through 32. Default is 1. **Note:** The Ethernet interface delivers its Ethernet Status (including Channel Status bits) during its input scan. Each channel's data transfer updates the Channels Status bits, so channels performance may be reduced if the Ethernet interface is configured to use an I/O Scan Set than runs more slowly than the PLC logic sweep. If the Ethernet interface is configured to use an inactive I/O Scan Set, the channel's Status bits will not be transferred and channel operations will not complete.

RS-232 Port (Station Manager) Parameters

These parameters are for the RS-232 Station Manager serial port.

Baud Rate	Data rate (bits per second) for the port. Choices are 1200**,** 2400**,** 4800**,** 9600**,** 19.2k, 38.4k, 57.6k, 115.2k**.**
Parity	Type of parity to be used for the port. Choices are None, Even, or Odd.
Flow Control	Type of flow control to be used for the port. Choices are None or Hardware. **Note:** The Hardware flow control is RTS/CTS crossed.
Stop Bits	The number of stop bits for serial communication. Choices are one or two.

Terminals Tab

This tab is displayed only when the Ethernet Interface's Variable Mode property is set to True. When Variable Mode is selected, the Ethernet status bits are referenced as I/O variables that are mapped to the Ethernet status bits on this configuration tab.

The use of I/O variables allows you to configure the Ethernet Interface without having to specify the reference addresses to use for the status information. Instead, you can directly associate variable names with the status bits. For more information, refer to "I/O Variables" in Chapter 7.

Advanced User Parameters

Advanced User Parameters (AUP) are additional configuration parameters for the Ethernet Interface. The default values are chosen to satisfy the vast majority of users. These default AUP values are usually modified only under exceptional circumstances by experienced users. The default AUP values for an Ethernet Interface may be changed by creating an AUP file, using the programming software to import the AUP file into the HW Configuration for that Ethernet Interface, and then storing the configuration to the PLC. For details on creating an AUP file, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.

Caution

The IEEE 802.3 standard strongly discourages the manual configuration of duplex mode for a port (as would be possible using Advanced User Parameters.) Before manually configuring duplex mode for a port using AUP, be sure that you know the characteristics of the link partner and are aware of the consequences of your selection. In the words of the IEEE standard: "Connecting incompatible DTE/MAU combinations such as full duplex mode DTE to a half duplex MAU, or a full-duplex station (DTE or MAU) to a repeater or other half duplex network, can lead to severe network performance degradation, increased collisions, ate collisions, CRC errors, and undetected data corruption."

Note: If speed and duplex mode of a port are forced using AUP, the switch will no longer perform automatic cable detection. This means that if you have the switch port connected to a switch or hub port you must use a crossover network cable. If you have the switch port connected to the uplink port on a switch or hub or if you have the switch port connected to another Ethernet device, you must use a normal network cable.

Verifying Proper Power-Up of the Ethernet Interface

After configuring the Interface, follow the procedure below to verify that the Ethernet Interface is operating correctly.

1. Turn power OFF to the CPU for 3–5 seconds, then turn the power back ON. This initiates a series of diagnostic tests. The EOK LED will blink indicating the progress of power-up.
2. The Ethernet LEDs will have the following pattern upon successful power-up. (For details on LED operation, see “CPU and Ethernet LEDs” in chapter 2.) At this time the Ethernet Interface is fully operational and on-line.

LED	***Ethernet Interface Online***
EOK	On
LAN	On, Off, or blinking, depending on network activity
STAT	On

If a problem is detected during power-up, the Ethernet Interface may not transition directly to the operational state. For diagnostics, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.

Pinging TCP/IP Ethernet Interfaces on the Network

PING (Packet InterNet Grouper) is a program used on TCP/IP networks to test reachability of destinations by sending them an ICMP echo request message and waiting for a reply. Most nodes on TCP/IP networks, including the PACSystems Ethernet Interface, implement a PING command.

You should ping each installed Ethernet Interface. When the Ethernet Interface responds to the ping, it verifies that the interface is operational and configured properly. Specifically it verifies that acceptable TCP/IP configuration information has been stored to the Interface.

Pinging the Interface from a UNIX Host or a PC Running TCP/IP Software

A PING command can be executed from a UNIX host or PC running TCP/IP or from another Ethernet Interface. When using a PC or UNIX host, you can refer to the documentation for the ping command, but in general all that is required is the IP address of the remote host as a parameter to the ping. For example, at the command prompt type:

```
ping  10.0.0.1
```

Determining if an IP Address is Already Being Used

Note: This method does not guarantee that an IP address is not duplicated. It will not detect a device that is configured with the same IP address if it is temporarily off the network.

It is very important not to duplicate IP addresses. To determine if another node on the network is using the same IP address:

1. Disconnect your Interface from the LAN.
2. Ping the disconnected Interface’s IP address. If you get an answer to the ping, the chosen IP address is already in use by another node. You must correct this situation by assigning unique IP addresses.

Chapter

5

CPU Operation

This chapter describes the operating modes of a PACSystems CPU and describes the tasks the CPU carries out during these modes. The following topics are discussed:

- CPU Sweep
- Program Scheduling Modes
- Window Modes
- Run/Stop Operations
- Flash Memory Operation
- Clocks and Timers
- System Security
- I/O System
- Power-Up and Power-Down Sequences

CPU Sweep

The application program in the CPU executes repeatedly until stopped by a command from the programmer, from another device, from the Run/Stop switch on the CPU module, or a fatal fault occurs. In addition to executing the application program, the CPU obtains data from input devices, sends data to output devices, performs internal housekeeping, performs communications tasks, and performs self-tests. This sequence of operations is called the ***sweep***.

The CPU sweep runs in one of three sweep modes:

Normal Sweep	In this mode, each sweep can consume a variable amount of time. The Logic Window is executed in its entirety each sweep. The Communications and Background Windows can be set to execute in Limited or Run-to-Completion mode.
Constant Sweep	In this mode, each sweep begins at a user-specified Constant Sweep time after the previous sweep began. The Logic Window is executed in its entirety each sweep. If there is sufficient time at the end of the sweep, the CPU alternates among the Communications and Background Windows, allowing them to execute until it is time for the next sweep to begin.
Constant Window	In this mode, each sweep can consume a variable amount of time. The Logic Window is executed in its entirety each sweep. The CPU alternates among the Communications and Background Windows, allowing them to execute for a time equal to the user-specified Constant Window timer.

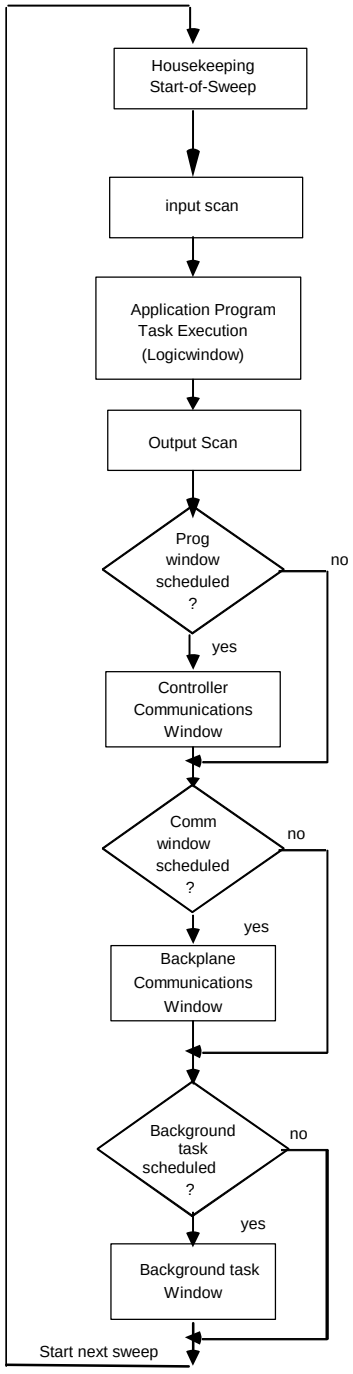
Note: The information presented above summarizes the different sweep modes. For additional information, refer to “CPU Sweep Modes” on page 5-6.

The CPU also operates in one of four Run/Stop Modes (for details, see “Run/Stop Operations” on page 5-10):

- Run/Outputs Enabled
- Run/Outputs Disabled
- Stop/IO Scan
- Stop/No IO

Parts of the CPU Sweep

There are seven major phases in a typical CPU sweep as shown in the following figure.



Parts of a Typical CPU Sweep

Major Phases in a Typical CPU Sweep

Phase	Activity
Housekeeping	The housekeeping portion of the sweep performs the tasks necessary to prepare for the start of the sweep. This includes updating %S bits, determining timer update values, determining the mode of the sweep (Stop or Run), and polling of expansion racks. Expansion racks are polled to determine if power has just been applied to an expansion rack. Once an expansion rack is recognized, then configuration of that rack and all of its modules are processed in the Controller Communications Window.
Input Scan	During the input scan, the CPU reads input data from the Genius Bus Controllers and input modules. If data has been received on an EGD page, the CPU copies the data for that page from the Ethernet interface to the appropriate reference memory. For details, see *TCP/IP Ethernet Communications for PACSystems,* GFK-2224 **Note:** The input scan is not performed if a program has an active Suspend I/O function on the previous sweep.
Application Program Task Execution (Logic Window)	The CPU solves the application program logic. It always starts with the first instruction in the program. It ends when the last instruction is executed. Solving the logic creates a new set of output data. For details on controlling the execution of programs, refer to chapter 6. Interrupt driven logic can execute during any phase of the sweep. For details, refer to chapter 6. A list of execution times for instructions can be found in Appendix A.
Output Scan	The CPU writes output data to bus controllers and output modules. The user program checksum is computed. During the output scan, the CPU sends output data to the Genius Bus Controllers and output modules. If the producer period of an EGD page has expired, the CPU copies the data for that page from the appropriate reference memory to the Ethernet interface. The output scan is completed when all output data has been sent. If the CPU is in Run mode and it is configured to perform a background checksum calculation, the background checksum is performed at the end of the output scan. The default setting for number of words to checksum each sweep is 16. If the words to checksum each sweep is set to zero, this processing is skipped. The background checksum helps ensure the integrity of the user logic while the CPU is in Run mode. The output scan is not performed if a program has an active Suspend I/O function on the current sweep.
Controller Communications Window	Services the onboard Ethernet and serial ports. In addition, reconfiguration of expansion racks and individual modules occurs during this portion of the sweep. The CPU always executes this window. The following items are serviced in this window: ■ Reconfiguration of expansion racks and individual modules. During the Controller Communications Window, highest priority is given to reconfiguration. Modules are reconfigured as needed, up to the total time allocated to this window. Several sweeps are required to complete reconfiguration of a module. ■ Communications activity involving the embedded Ethernet port and the two CPU's serial ports Time and execution of the Controller Communications Window can be configured using the programming software. It can also be dynamically controlled from the user program using Service Request function #3. The window time can be set to a value from 0 to 255 milliseconds (default is 10 milliseconds). Note that if the Controller Communications Window is set to 0, there are two alternate ways to open the window: perform a power-cycle without the battery, or go to Stop mode.

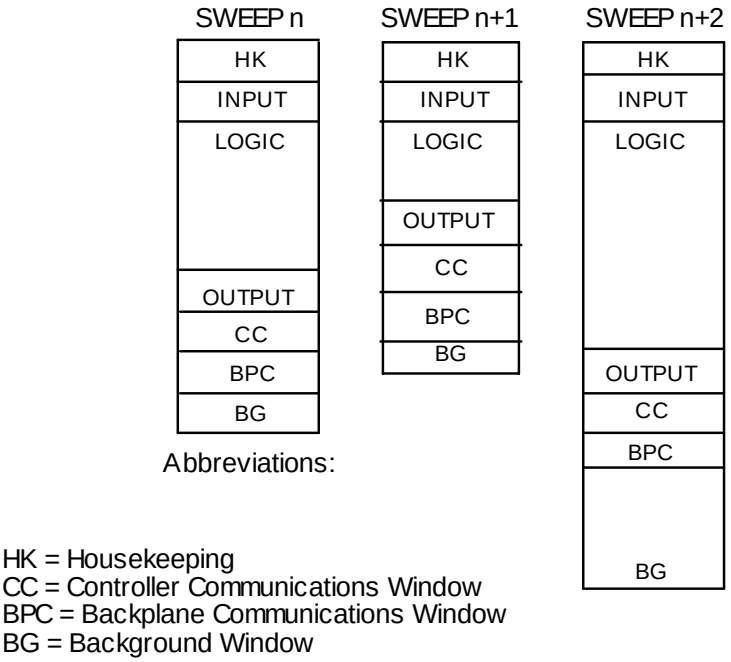
Phase	*Activity*
Backplane Communications Window	Communications with intelligent devices occur during this window. The rack-based Ethernet Interface module communicates in the Backplane Communications window. During this part of the sweep the CPU communicates with intelligent modules such as the Genius Bus Controller and TCP/IP Ethernet modules. In this window, the CPU completes any previously unfinished request before executing any pending requests in the queue. When the time allocated for the window expires, processing stops. The Backplane Communications Window defaults to Complete (Run to Completion) mode. This means that all currently pending requests on all intelligent option modules are processed every sweep. This window can also run in Limited mode, in which the maximum time allocated for the window per scan is specified. The mode and time limit can be configured and stored to the CPU, or it can be dynamically controlled from the user program using Service Request function #4. The Backplane Communications Window time can be set to a value from 0 to 255ms (default is 255ms). This allows communications functions to be skipped during certain time-critical sweeps.
Background Window	CPU self-tests occur in this window. A CPU self-test is performed in this window. Included in this self-test is a verification of the checksum for the CPU operating system software. The Background Window time defaults to 0 milliseconds. A different value can be configured and stored to the CPU, or it can be changed online using the programming software. Time and execution of the Background Window can also be dynamically controlled from the user program using Service Request function #5. This allows background functions to be skipped during certain time-critical sweeps.

CPU Sweep Modes

Normal Sweep Mode

In Normal Sweep mode, each sweep can consume a variable amount of time. The Logic window is executed in its entirety each sweep. The Communications windows can be set to execute in a Limited or Run-to-Completion mode. Normal Sweep is the most common sweep mode used for control system applications.

The following figure illustrates three successive CPU sweeps in Normal Sweep mode. Note that the total sweep times may vary due to sweep-to-sweep variations in the Logic window, Communications windows, and Background window.



Typical Sweeps in Normal Sweep Mode

Constant Sweep Mode

In Constant Sweep mode, each sweep begins at a specified Constant Sweep time after the previous sweep began. The Logic Window is executed in its entirety each sweep. If there is sufficient time at the end of the sweep, the CPU alternates among the Controller Communications, Backplane Communications, and Background Windows, allowing them to execute until it is time for the next sweep to begin. Some or all of the Communications and Background Windows may not be executed. The Communications and Background Windows terminate when the overall CPU sweep time has reached the value specified as the Constant Sweep time.

One reason for using Constant Sweep mode is to ensure that I/O data are updated at constant intervals.

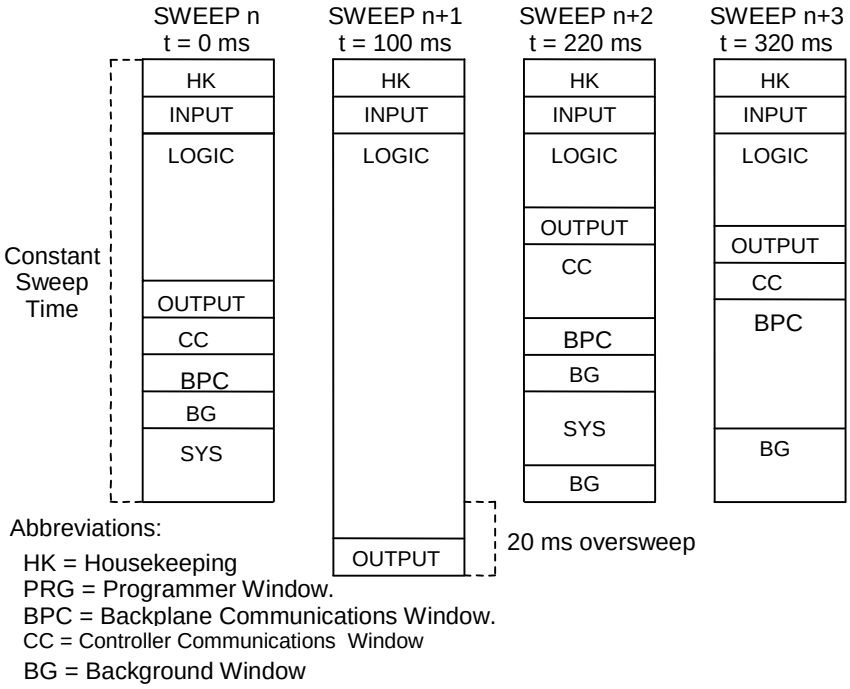
The value of the Constant Sweep timer can be configured to be any value from 5 to 2550 milliseconds. The Constant Sweep timer value may also be set and Constant

Sweep mode may be enabled or disabled by the programming software or by the user program using Service Request function #1. The Constant Sweep timer has no default value; a timer value must be set prior to or at the same time Constant Sweep mode is enabled.

The Ethernet Global data page configured for either consumption or production can add up to 1 millisecond to the sweep time. This sweep impact should be taken into account when configuring the CPU constant sweep mode and setting the CPU watchdog timeout.

If the sweep exceeds the Constant Sweep time in a given sweep, the CPU places an oversweep alarm in the CPU fault table and sets the OV_SWP (%SA0002) status reference at the beginning of the next sweep. Additional sweep time due to an oversweep condition in a given sweep does not affect the time given to the next sweep.

The following figure illustrates four successive sweeps in Constant Sweep mode with a Constant Sweep time of 100 milliseconds. Note that the total sweep time is constant, but an oversweep may occur due to the Logic Window taking longer than normal.



Typical Sweeps in Constant Sweep Mode

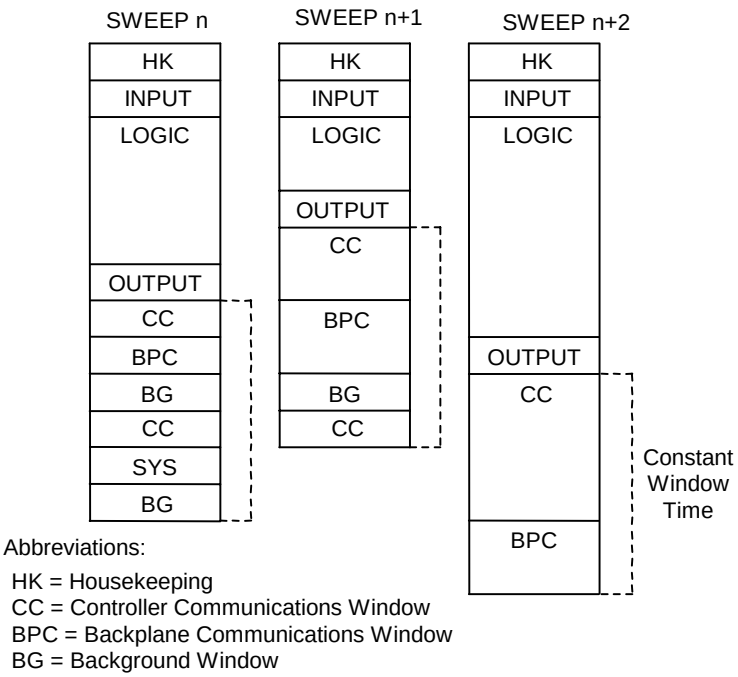
Constant Window Mode

In Constant Window mode, each sweep can consume a variable amount of time. The Logic Window is executed in its entirety each sweep. The CPU alternates among the three windows, allowing them execute for a time equal to the value set for the Constant Window timer. The overall CPU sweep time is equal to the time required to execute the Housekeeping, Input Scan, Logic Window, and Output Scan phases of the sweep plus the value of the Constant Window timer. This time may vary due to sweep-to-sweep variances in the execution time of the Logic Window.

An application that requires a certain amount of time between the Output Scan and the Input Scan, permitting inputs to settle after receiving output data from the program, would be ideal for Constant Window mode.

The value of the Constant Window timer can be configured to be any value from 3 to 255 milliseconds. The Constant Window timer value may also be set by the programming software or by the user program using Service Request functions #3, #4, and #5.

The following figure illustrates three successive sweeps in Constant Window mode. Note that the total sweep times may vary due to sweep-to-sweep variations in the Logic Window, but the time given to the Communications and Background Windows is constant. Some of the Communications or Background Windows may be skipped, suspended, or run multiple times based on the Constant Window time.



Typical Sweeps in Constant Window Mode

Program Scheduling Modes

The CPU supports one program scheduling mode, the Ordered mode. An ordered program is executed in its entirety once per sweep in the Logic Window.

Window Modes

The previous section describes the phases of a typical CPU sweep. The Controller Communications, Backplane Communications, and Background windows can be run in various modes, based on the CPU sweep mode. (CPU sweep modes are described in detail on page 5-6.) The following three window modes are available:

Run-to-Completion	In Run-to-Completion mode, all requests made when the window has started are serviced. When all pending requests in the given window have completed, the CPU transitions to the next phase of the sweep. (This does not apply to the Background window because it does not process requests.)
Constant	In Constant Window mode, the total amount of time that the Controller Communications window, Backplane Communications window, and Background window run is fixed. If the time expires while in the middle of servicing a request, these windows are closed, and communications will be resumed the next sweep. If no requests are pending in this window, the CPU cycles through these windows the specified amount of time polling for further requests. If any window is put in constant window mode, all are in constant window mode.
Limited	In Limited mode, the maximum time that the window runs is fixed. If time expires while in the middle of servicing a request, the window is closed, and communications will be resumed the next time that the given window is run. If no requests are pending in this window, the CPU proceeds to the next phase of the sweep.

Data Coherency in Communications Windows

When running in Constant or Limited Window mode, the Controller and Backplane Communications Windows may be terminated early in all CPU sweep modes. If an external device, such as CIMPLICITY HMI, is transferring a block of data, the coherency of the data block may be disrupted if the communications window is terminated prior to completing the request. The request will complete during the next sweep; however, part of the data will have resulted from one sweep and the remainder will be from the following sweep. When the CPU is in Normal Sweep mode and the Communications Window is in Run-to-Completion mode, the data coherency problem described above does not exist.

Note: External devices that communicate to the CPU while it is stopped will read information as it was left in its last state. This may be misleading to operators viewing an HMI system that does not indicate CPU Run/Stop state. Process graphics will often indicate everything is still operating normally.

Also, note that non-retentive outputs do not clear until the CPU is changed from Stop to Run.

Run/Stop Operations

The PACSystems CPUs support four run/stop modes of operation. You can change these modes in the following ways: the Run/Stop switch, configuration from the programming software, LD function blocks, and system calls from C applications. Switching to and from various modes can be restricted based on privilege levels, position of the Run/Stop switch, passwords, etc.

Mode	*Operation*
Run/Outputs Enabled	The CPU runs user programs and continually scans inputs and updates physical outputs, including Genius and Ethernet outputs. The Controller and Backplane Communications Windows are run in Limited, Run-to-Completion, or Constant mode.
Run/Outputs Disabled	The CPU runs user programs and continually scans inputs, but updates to physical outputs, including Genius and Field Control, are not performed. Physical outputs are held in their configured default state in this mode. The Controller and Backplane Communications Windows are run in Limited, Run-to-Completion, or Constant mode.
Stop/IO Scan Enabled	The CPU does not run user programs, but the inputs and outputs are scanned. The Controller and Backplane Communications Windows are run in Run-to-Completion mode. The Background Window is limited to 10 ms.
Stop/IO Scan Disabled	The CPU does not run user programs, and the inputs and outputs are not scanned. The Controller and Backplane Communications Windows are run in a Run-to-Completion mode. The Background Window is limited to 10 ms. **Note:** Stop mode I/O scanning is always disabled for redundancy CPUs.

Note: You cannot add to the size of %P and %L reference tables in Run Mode unless the %P and %L references are the first of their type in the block being stored or the block being stored is a totally new block.

CPU Stop Modes

The CPU has two modes of operation while it is in Stop mode:

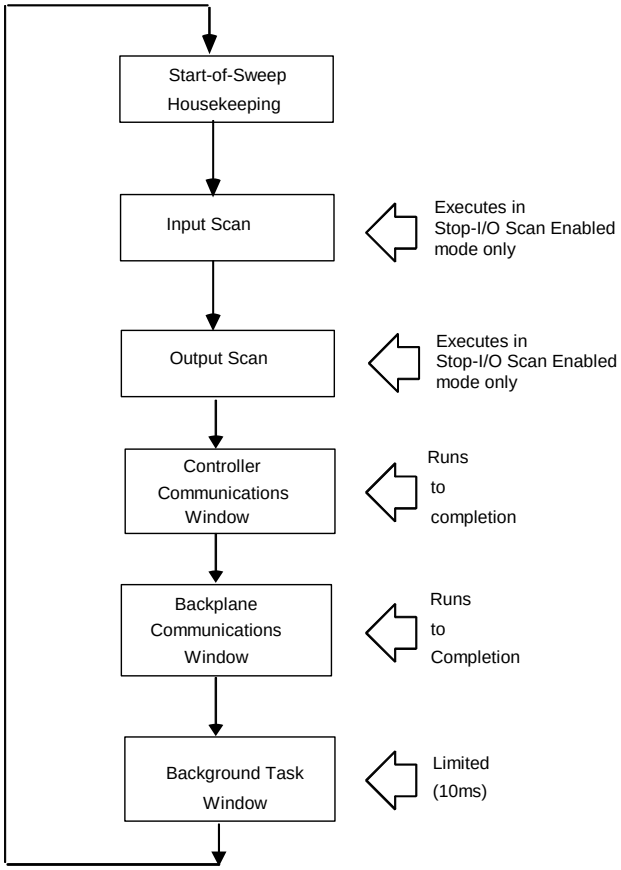
- I/O Scan Enabled - the Input and Output scans are performed each sweep
- I/O Scan Disabled - the Input and Output scans are skipped

When the CPU is in Stop mode, it does not execute the application program. You can configure whether the I/O is scanned during Stop mode. Communications with the programmer and intelligent option modules continue in Stop mode. Also, bus receiver module polling and rack reconfiguration continue in Stop mode.

In both Stop modes, the Controller Communications and Backplane Communications windows run in Run-to-Completion mode and the Background window runs in Limited mode with a 10 millisecond limit.

The number of last scans can be configured in the hardware configuration. Last scans are completed after the CPU has received an indication that a transition from Run to Stop or Stop Faulted mode should occur. The default is 0.

SVCREQ13 can be used in the application program to stop the CPU after a specified number of scans. All I/O will go to their configured default states, and a diagnostic message will be placed in the CPU Fault Table.



CPU Sweep in Stop- I/O Disabled and Stop- I/O Enabled Modes

Stop-to-Run Mode Transition

The CPU performs the following operations on Stop-to-Run transition:

- Validation of sweep mode and program scheduling mode selections
- Validation of references used by programs with the actual configured sizes
- Re-initialization of data areas for external blocks and standalone C programs
- Clearing of non-retentive memory

Run/Stop Mode Switch Operation

The Run/Stop mode switch has three positions:

Switch Position	*CPU and Sweep Mode*	*Memory Protection*
Run I/O Enable	The CPU runs with I/O sweep enabled.	User program memory is read only.
Run Output Disable	The CPU runs with I/O sweep disabled.	User program memory is read only.
Stop	The CPU is not allowed to go into Run mode.	User program memory can be written.

The Run/Mode switch can be disabled in the programming software HWC. The switch's memory protection function can be disabled separately in HWC. The Run/Mode switch is enabled by default. The memory protection functionality is disabled by default.

The Read Switch Position (Switch_Pos) function allows the logic to read the current position of the Run/Stop switch, as well as the mode for which the switch is configured. For details, refer to chapter 8.

Flash Memory Operation

The CPU stores the current configuration and application in battery-backed RAM. With PACSystems, you also have the option to store the Logic, Hardware Configuration, and Reference Data into resident flash memory. The PACSystems CPU provides enough flash memory to hold all of user space (10MB), all reference tables that aren't counted against user space, and any overhead required. For details on which items count against user memory space, please refer to appendix B.

By default, the CPU reads program logic and configuration, and reference table data from RAM at powerup. However, logic/configuration and reference tables can each be configured to always read from flash or conditionally read from flash. To configure these parameters in the programming software, select the CPU's Settings tab in Hardware Configuration.

If conditional flash is selected as the power-up source, the CPU powers up from flash only if a corrupted memory condition is detected or if there is no battery attached during the power cycle. Conditional powerup from flash is recommended, because data in flash is non-volatile and does not require a battery to maintain its data.

If logic/configuration and/or reference tables are configured for conditional powerup from flash, these items are restored from flash to battery-backed RAM when user memory is corrupted or was not preserved (i.e. no battery). If logic/configuration and/or reference memory are configured for conditional powerup from flash and user memory has been preserved, no flash operation will occur.

If logic/configuration and/or reference tables are configured to always power up from flash, these items are restored from flash to battery-backed RAM regardless of the state of the user memory.

Note: If ***any*** component (logic/configuration or reference tables) is read from flash, OEM-mode and passwords are also read from flash.

In addition to configuring where the CPU obtains logic, configuration, and data during powerup, the programming software provides the following flash operations:

- Write a copy of the current configuration, application program, and reference tables (excluding overrides) to flash memory. Note that a write-to-flash operation causes all components to be stored to flash.
- Read a previously stored configuration and application program, and/or reference table values from flash into RAM.
- Verify that flash and RAM contain identical data.
- Clear flash contents.

Flash read and write operations copy the contents of flash memory or RAM as individual files. The programming software displays the progress of the copy operation and allows you to cancel a flash read or write operation during the copy process instead of waiting for the entire transfer process to complete. The entire user memory image must be successfully transferred for the flash copy to be considered successful. If an entire write-to-flash transfer is not completed due to canceling, power cycle, or some other intervention, the CPU will clear flash memory. Similarly, if a read-from-flash transfer is interrupted, RAM will be cleared.

Logic/Configuration Source and CPU Operating Mode at Power-up

Flash and user memory can contain different values for the Logic/Configuration Power-up Source parameter. The following tables summarize how these settings determine the logic/configuration source after a power cycle. CPU mode is affected by the Power-up Mode, Run/Stop Switch and Stop-Mode I/O Scanning parameters, Run/Stop mode switch position, and the power down mode as shown in the tables on page 5-15.

Before Power Cycle		***After Power Cycle***	
Logic/Configuration Power-up Source in Flash	***Logic/Configuration Power-up Source in RAM***	***Origin of Logic/Configuration***	***CPU Mode***
Always Flash	Memory not preserved (i.e. no battery or memory corrupted)	Flash	See “CPU Mode when Memory Not Preserved/ Power-up Source is Flash” on page 5-15.
Always Flash	No configuration in RAM, memory preserved	Flash	See” Memory Preserved” on page 5-15.
Always Flash	Always Flash	Flash	
Always Flash	Conditional Flash	Flash	
Always Flash	Always RAM	Flash	
Conditional Flash	Memory not preserved (i.e. no battery or memory corrupted)	Flash	See “CPU Mode when Memory Not Preserved/ Power-up Source is Flash” on page 5-15.
Conditional Flash	No configuration in RAM, memory preserved	Uses default logic/configuration	Stop Disabled
Conditional Flash	Always Flash	RAM	See ”CPU Mode when Memory Preserved” on page 5-15.
Conditional Flash	Conditional Flash	RAM	
Conditional Flash	Always RAM	RAM	
Always RAM	Memory not preserved (i.e. no battery or memory corrupted)	Uses default logic/configuration	Stop Disabled
Always RAM	No configuration in RAM, memory preserved	Uses default logic/configuration	Stop Disabled
Always RAM	Always Flash	Flash	See ”CPU Mode when Memory Preserved” on page 5-15.
Always RAM	Conditional Flash	RAM	
Always RAM	Always RAM	RAM	
No Configuration in Flash	Memory not preserved (i.e. no battery or memory corrupted)	Uses default logic/configuration	Stop Disabled
No Configuration in Flash	No configuration in RAM, memory preserved	Uses default logic/configuration	Stop Disabled
No Configuration in Flash	Always Flash	RAM	See ”CPU Mode when Memory Preserved” on page 5-15.
No Configuration in Flash	Conditional Flash	RAM	
No Configuration in Flash	Always RAM	RAM	

CPU Mode when Memory Not Preserved/Power-up Source is Flash

Configuration Parameters		Run/Stop Switch Position	CPU Mode
Power-up Mode	Run/Stop Switch		
Run	Enabled	Stop	Stop Disabled
Run	Enabled	Run Disabled	Run Disabled
Run	Enabled	Run Enabled	Run Enabled
Run	Disabled	N/A	Run Disabled
Stop	N/A	N/A	Stop Disabled
Last	Enabled	Stop	Stop Disabled
Last	Enabled	Run Disabled	Run Disabled
Last	Enabled	Run Enabled	Run Disabled
Last	Disabled	N/A	Run Disabled

CPU Mode when Memory Preserved

Configuration Parameters			Run/Stop Switch Position	Power Down Mode	CPU Mode
Power-up Mode	Run/Stop Switch	Stop-Mode I/O Scanning			
Run	Enabled	Enabled	Stop	N/A	Stop Enabled
Run	Enabled	Disabled	Stop	N/A	Stop Disabled
Run	Enabled	N/A	Run Disabled	N/A	Run Disabled
Run	Enabled	N/A	Run Enabled	N/A	Run Enabled
Run	Disabled	N/A	N/A	N/A	Run Enabled
Stop	N/A	Enabled	N/A	N/A	Stop Enabled
Stop	N/A	Disabled	N/A	N/A	Stop Disabled
Last	Enabled	Enabled	Stop	Stop Disabled	Stop Disabled
Last	Enabled	Enabled	Stop	Stop Enabled	Stop Enabled
Last	Enabled	Enabled	Stop	Run Disabled	Stop Enabled
Last	Enabled	Enabled	Stop	Run Enabled	Stop Enabled
Last	Enabled	Disabled	Stop	N/A	Stop Disabled
Last	Enabled	N/A	Run Disabled	Stop Disabled	Stop Disabled
Last	Enabled	Enabled	Run Disabled	Stop Enabled	Stop Enabled
Last	Enabled	Disabled	Run Disabled	Stop Enabled	Stop Disabled
Last	Enabled	N/A	Run Disabled	Run Disabled	Run Disabled
Last	Enabled	N/A	Run Disabled	Run Enabled	Run Disabled
Last	Enabled	N/A	Run Enabled	Stop Disabled	Stop Disabled
Last	Enabled	Enabled	Run Enabled	Stop Enabled	Stop Enabled
Last	Enabled	Disabled	Run Enabled	Stop Enabled	Stop Disabled
Last	Enabled	N/A	Run Enabled	Run Disabled	Run Disabled
Last	Enabled	N/A	Run Enabled	Run Enabled	Run Enabled
Last	Disabled	N/A	N/A	Stop Disabled	Stop Disabled
Last	Disabled	Enabled	N/A	Stop Enabled	Stop Enabled
Last	Disabled	Disabled	N/A	Stop Enabled	Stop Disabled
Last	Disabled	N/A	N/A	Run Disabled	Run Disabled
Last	Disabled	N/A	N/A	Run Enabled	Run Enabled

Clocks and Timers

Clocks and timers provided by the CPU include an elapsed time clock, a time-of-day clock, and software and hardware watchdog timers.

For information on timer functions and timed contacts provided by the CPU instruction set, see “Timers and Counters” in chapter 8.

Elapsed Time Clock

The elapsed time clock tracks the time elapsed since the CPU powered on. The clock is not retentive across a power failure; it restarts on each power-up. This seconds count rolls over (seconds count returns to zero) approximately 100 years after the clock begins timing.

Because the elapsed time clock provides the base for system software operations and timer function blocks, it may not be reset from the user program or the programmer. However, the application program can read the current value of the elapsed time clock by using Service Request #16 or Service Request #50, which provides higher resolution.

Time-of-Day Clock

A hardware time-of-day clock maintains the time of day (TOD) in the CPU. The time-of-day clock maintains the following seven time functions:

- Year (two digits)
- Month
- Day of month
- Hour
- Minute
- Second
- Day of week

The TOD clock is battery-backed and maintains its present state across a power failure. The time-of-day clock handles month-to-month and year-to-year transitions and automatically compensates for leap years through year 2036.

You can read and set the hardware TOD time and date through the application program using Service Request function #7. For details, see chapter 10.

High-Resolution Time of Day Software Clock

A high-resolution software TOD clock is implemented in firmware to provide nanoseconds resolution. When the high-resolution software TOD clock is set, the hardware TOD clock is set with the YYYY: Mon: Day: Hr: Min: Sec fields in the POSIX time, the RTC is read, and the delta between the POSIX time and the value read from the RTC is computed and saved. Thus, if 1-second resolution is desired the hardware TOD clock is read. Otherwise, the high-resolution software TOD clock is read to provide greater resolution. When the latter occurs, the hardware RTC is read and the saved delta added to the value read.

When the SNTP Time Transfer feature is implemented, all SNTP time updates received at the CPU shall update the high-resolution software TOD clock.

Synchronizing the High-resolution Time of Day Clock to an SNTP Network Time Server

In an SNTP system, a computer on the network (called an SNTP server) sends out a periodic timing message to all SNTP-capable Ethernet Interfaces on the network, which synchronize their internal clocks with this SNTP timing message. If SNTP is used to perform network time synchronization, the timestamp information typically has ±10 millisecond accuracy between PLCs on the same network.

Synchronizing the CPU TOD clock to an SNTP server allows you to set a consistent time across multiple systems. Once the CPU TOD clock is synchronized with the SNTP time, all produced EGD exchanges will use the CPU's TOD for the time stamp.

The CPU TOD clock is set with accuracy within ±2 ms of the SNTP time stamp.

TOD clock synchronization is enabled on an Ethernet module by the advanced user parameter (AUP), *ncpu_sync*. The CPU must also use a COMMREQ in user logic to select an Ethernet module as the time master. For additional information, refer to "Timestamping of Ethernet Global Data Exchanges" in chapter 4 of *TCP/IP Communications for PACSystems,* GFK-2224.

Watchdog Timer

Software Watchdog Timer

A software watchdog timer in the CPU is designed to detect “failure to complete sweep” conditions. The timer value for the software watchdog timer is set by using the programming software. The allowable range for this timer is 10 to 2550 milliseconds; the default value is 200 milliseconds. The software watchdog timer always starts from zero at the beginning of each sweep.

The software watchdog timer is useful in detecting abnormal operation of the application program that prevents the CPU sweep from completing within the user-specified time. Examples of such abnormal application program conditions are as follows:

- Excessive recursive calling of a block
- Excessive looping (large loop count or large amounts of execution time for each iteration)
- Infinite execution loop

When selecting a software watchdog value, always set the value higher than the longest expected sweep time to prevent accidental expiration. For Constant Sweep mode, allowance for oversweep conditions should be considered when selecting the software watchdog timer value.

The watchdog timer continues during interrupt execution. Queuing of interrupts within a single sweep may cause watchdog timer expiration.

If the software watchdog timeout value is exceeded, the OK LED blinks, and the CPU goes to Stop/Halt mode. Certain functions, however, are still possible. A fault is placed in the CPU fault table, and outputs go to their default state. The CPU will only communicate with the programmer attached through the embedded Ethernet interface; no other communications or operations are possible. To recover, power must be cycled on the rack or backplane containing the CPU.

To extend the current sweep beyond the software watchdog timer value, the application program may restart the software watchdog timer using Service Request function #8. However, the software watchdog timer value may only be changed from the configuration software.

Note that Service Request Function #8 does not reset the output scan timer implemented on the Genius Bus Controller.

Hardware Watchdog Timer

A backup circuit provides additional protection for the CPU. If this backup circuit activates, the CPU is immediately placed in Reset mode. Outputs go to their default state and no communications of any form are possible, and the CPU will halt. To recover, power must be cycled.

Note: Fatal Fault Retries is not supported by PACSystems.

System Security

The PACSystems CPU supports the following two types of system security:

- Passwords/privilege levels
- OEM protection

Passwords and Privilege Levels

Passwords are a configurable feature of the PACSystems CPU. Their use is optional and can be set up using the programming software. Passwords provide different levels of access privilege for the CPU when the programmer is Online. Passwords are not used if the programmer is in Offline mode.

The default state is no password protection. Each privilege level in the CPU may have a unique password; however, the same password can be used for more than one level. Passwords are one to seven ASCII characters in length. Passwords can be changed only through the programming software.

After passwords have been set up, access to the CPU via any communications path is restricted from the levels at which the passwords are set, unless the proper password has been entered. Once a password has successfully been accepted, access to the privilege level requested and below is granted (for example, providing the password for level 3 allows access to functions at levels 1, 2, and 3).

Note: The Run Mode switch on the CPU overrides password protection. Even though the programmer may not be able to switch between Run and Stop mode, the switch on the CPU can do so.

Privilege Levels

Priv Level	***Password***	***Access Description***
4	Yes	Write to configuration or logic. Configuration may only be written in Stop mode; logic may be written in Stop or Run mode. Set or delete passwords for any level. Note: This is the default privilege for a connection to the CPU if no passwords are defined.
3	Yes	Write to configuration or logic when the CPU is in Stop mode, including word-for-word changes, addition/deletion of program logic, and the overriding of discrete I/O.
2	Yes	Write to any data memory. This does not include overriding discrete I/O. The CPU can be started or stopped. CPU and I/O fault tables can be cleared.
1	Yes	Read any CPU data, except for passwords. This includes reading fault tables, performing datagrams, verifying logic/configuration, loading program and configuration, etc. from the CPU. None of this data may be changed. At this level, transition to Run mode from the programmer is not allowed.

Protection Level Request from Programmer

Upon connection to the CPU, the programmer requests the CPU to move to the highest non-protected level.

The programmer requests a privilege level change by supplying the new privilege level and the password for that level. If the password sent by the programmer does not agree with the password stored in the CPU's password access table for the requested level, the privilege level change is denied and a fault is logged in the CPU fault table. The current privilege level is maintained, and no change occurs. A request to change to a privilege level that is not password protected is made by supplying the new level and a null password. A privilege change may be to a lower level as well as to a higher level.

Disabling Passwords

The use of password protection is optional. If you want to prevent the use of password protection, passwords can be disabled using the programming software.

Note: To enable passwords after they have been disabled, the CPU must be power-cycled with the battery removed.

Password protection prevents firmware upgrades. Prior to attempting a firmware upgrade, disable password protection, then enable it after the upgrade.

OEM Protection

OEM protection is similar to the passwords and privilege levels. However, OEM protection provides a higher level of security. The OEM protection feature is enabled/disabled using a 1 to 7 character password. When OEM protection is enabled, all read and write access to the CPU program and configuration is prohibited.

Protection for OEMs' investment in software is provided in the form of a special password known as the *OEM key*. When the OEM key has been given a non-NULL value, the CPU may be placed in a mode in which reads, writes, and verification of the logic and/or configuration are prohibited. This allows a third-party OEM to create Control Programs for the CPU and then set the OEM-locked mode, which prevents the end user from reading or modifying the program.

Note: OEM protection prevents firmware upgrades to the flash memory. To upgrade the firmware, you must first disable OEM protection, then enable it again after the upgrade.

PACSystems I/O System

The PACSystems I/O system provides the interface between the CPU and other devices. The PACSystems I/O system supports:

- I/O and Intelligent option modules.
- Ethernet Interface
- Motion modules (RX3i)
- The Genius I/O system (RX7i). A Genius I/O Bus Controller (GBC) module provides the interface between the RX7i CPU and a Genius I/O bus.

I/O Configuration

Module Identification

In addition to the catalog number, the programming software stores a Module ID for each configured module in the hardware configuration that it delivers to the CPU. The CPU uses the Module ID to determine how to communicate with a given module.

When the hardware configuration is downloaded to the CPU (and during subsequent power-ups), the CPU compares the Module IDs stored by the programmer with the IDs of the modules physically present in the system. If the Module IDs do not match, a System Configuration Mismatch fault will be generated.

Because I/O modules of similar type may share the same Module ID, it is possible to download a configuration containing a module catalog number that does not match the module that is physically present in the slot without generating a System Configuration Mismatch.

Certain discrete modules with both reference memory inputs and reference memory outputs will experience invalid I/O transfer if incorrect configuration is stored from a similar mixed I/O module. No fault or error condition will be detected during configuration store and the module will be operational, although not in the manner described by configuration.

For example, a configuration swap between the IC693MDL754 output module and IC693MDL660 input module will not be detected as a configuration mismatch, but I/O data transfer between the module and the CPU reference memory will be invalid. If the input module (MDL660) is sent the configuration of the output module (MDL754) with the following parameters: Reference Address: %Q601
Module Status Reference: %I33
Hold Last State Enable

It will receive inputs at the module status reference %I33 and the status of the module will be received at %Q601.

If the output module is sent the configuration of the input module with the following parameters: Reference Address: %I601
Input Filter: Enable
Digital Filter Settings Reference: %I65

It will output values at the digital filter settings reference %I65 and the status of the module will be received at %I601.

Default Conditions for I/O Modules

Interrupts

Some input modules can be configured to send an interrupt to the application program. By default, this interrupt is disabled and the input filter is set to slow. If changed by the programming software, the new settings are applied when the configuration is stored and during subsequent power-cycles.

Outputs

Some output modules have a configurable output default mode that can be specified as either Off or Hold Last State. If a module does not have a configurable output default mode, its output default mode is Off. The selected action applies when the CPU transitions from Run/Enabled to Run/Disabled or Stop mode, or experiences a fatal fault.

At power-up, Series 90-30 discrete output modules default to all outputs off. They will retain this default condition until the first output scan from the PLC. Analog output modules can be configured with a jumper located on the module's removable terminal block to either default to zero or retain their last state.

Inputs

Input modules that have a configurable input default mode can be configured to Hold Last State or to set inputs to 0. If a module does not have a configurable input default mode, its input default mode is Off. The selected action applies when the CPU transitions from Run/Enabled to Run/Disabled or Stop mode, or experiences a fatal fault.

For details on the powerup and stop mode behavior of other modules, refer to the documentation for that module.

Multiple I/O Scan Sets

Up to 32 I/O scan sets can be defined for a PACSystems CPU. A scan set is a group of I/O modules that can be assigned a unique scan rate. A given I/O module can belong to one scan set. By default, all I/O modules are assigned to scan set 1, which is scanned every sweep.

For some applications, the CPU logic does not need to have the I/O information every sweep. The I/O scan set feature allows the scanning of I/O points to be more closely scheduled with their use in user logic programs. If you have a large number of I/O modules, you may be able to significantly reduce scan time by staggering the scanning of those modules.

A disadvantage of placing all modules into different scan sets appears when the CPU is transitioning from Stop to Run. In that case, scan sets with a programmed delay are not scanned on the first sweep. These modules' outputs are not enabled until the new data has been scanned to them, perhaps many scans later. Therefore there is a period of time during which the user logic is executing and some modules' outputs are disabled. During that time, outputs of those modules are in the module's stop-mode state. Stop-mode behavior is module-dependent. Some modules zero their outputs, some hold their last scanned state (if any), and some force their outputs to a configured default value. When the module's outputs are enabled, the module uses

the last scanned value, which will either be zero or the contents of the register the module uses to hold the corresponding output values from the reference tables.

Genius I/O

The Genius Bus Controller (GBC) controls a single Genius I/O bus. Any type of Genius I/O device may be attached to the bus.

In the I/O fault table, the rack, slot, bus, module, and I/O point number are given for a fault. Bus number one refers to the bus on the single-channel GBC.

Genius I/O Configuration

The programming software can configure a subset of the parameters associated with Genius I/O blocks.

Genius I/O blocks have a number of parameters that can be set using the Genius I/O Hand-Held Monitor. These parameter values are stored in EEPROM in the block itself. The serial bus address (SBA) and baud rate must be set using the Genius I/O Hand-Held Monitor. For specific information on Genius I/O block types, configuration, and setup, refer to the *Genius I/O System User's Manuals*, GEK-90486-1 and -2.

Through the COMMREQ function block, the application program can request the GBC to change any default condition on a specific block. However, the block only accepts this change if it is not in Config Protect mode. If Config Protect mode is set, only the Hand-Held Monitor can be used to change the defaults. The format of the COMMREQ function block for Genius I/O is described in the *Series 90-70 Genius Bus Controller User's Manual*, GFK-2017 and the *Series 90-30 Genius Bus Controller User's Manual,* GFK-1034.

Genius I/O Data Mapping

Genius I/O discrete inputs and outputs are stored as bits in the CPU Bit Cache memory. Genius I/O analog data is stored in the application RAM allocated for that purpose (%AI and %AQ). Analog data is always stored one channel per one word (16 bit).

An analog grouped module consumes (in the input and output data memories) only the amount of data space required for the actual inputs and outputs. For example, the Genius I/O 115 VAC Grouped Analog Block, IC660CBA100, has four inputs and two outputs. It consumes four words of Analog Input memory (%AI) and two words of Analog Output memory.

A discrete grouped module, each point of which is configurable with the Hand-Held Monitor (HHM) to be input, output, or output with feedback, consumes an amount in both discrete input memory (%I) and discrete output memory (%Q) equal to its physical size. Therefore, the eight-point Discrete Grouped Block (IC660CBD100) requires eight bits in the %I memory and eight bits in the %Q memory, regardless of how each point on the block is configured.

Analog Grouped Block

The six-channel Analog Grouped block contains four analog input channels and two analog output channels. When this block gets its turn on the Genius I/O Bus, it broadcasts the data for all four input channels in one broadcast control message. Then, when the GBC gets its turn, it sends the data for both output channels to the block in a directed control message.

Low-Level Analog Blocks

Unlike the Analog Grouped block, the low-level analog blocks, such as the Thermocouple and RTD blocks, are input-only blocks. All have six channels.

Genius Global Data Communications

The PACSystems RX7i supports the sharing of data among multiple control systems that share a common Genius I/O bus. This mechanism provides a means for the automatic and repeated transfer of %G, %I, %Q, %AI, %AQ, %R, and %W data. No special application programming is required to use global data since it is integrated into the I/O scan. All GE Fanuc controllers that have Genius I/O capability can send global data to an RX7i and can receive data from an RX7i. The programming software is used to configure the receiving and transmitting of global data on a Genius I/O bus.

Note: Genius global data communications do not continue to operate when the RX7i CPU is in Stop-I/O Scan Disabled mode. However, if the CPU is in Stop-I/O Scan Enabled mode, Genius global data communications continue to operate.

I/O System Diagnostic Data Collection

Diagnostic data in a PACSystems I/O system is obtained in either of the following two ways:

- If an I/O module has an associated bus controller, the bus controller provides the module's diagnostic data for the CPU. For details on GBC faults, see "PACSystems Handling of GBC Faults" on page 5-25.
- For I/O modules not interfaced through a bus controller, the CPU's I/O Scanner subsystem generates the diagnostic bits based on data provided by the module.

The diagnostic bits are derived from the diagnostic data sent from the I/O modules to their I/O controllers (CPU or bus controller). Diagnostic bits indicate the current fault status of the associated module. Bits are set when faults occur and are cleared when faults are cleared.

Diagnostic data is not maintained for non-GE Fanuc modules. The application program must use the BUS Read function blocks to access diagnostic information provided by those boards.

Discrete I/O Diagnostic Information

The CPU maintains diagnostic information for each discrete I/O point. Two memory blocks are allocated in application RAM for discrete diagnostic data, one for %I memory and one for %Q memory. One bit of diagnostic memory is associated with each I/O point. This bit indicates the validity of the associated I/O data. Each discrete point has a fault reference that can be interrogated using two special contacts: a fault

contact (-[F]-) and a no-fault contact (-[NF]-). The CPU collects this fault data if enabled to do so by the programming software. The following table shows the state of the fault and no-fault contacts.

Condition	*[FAULT]*	*[NOFLT]*
Fault Present	ON	OFF
Fault Absent	OFF	ON

Analog I/O Diagnostic Data

Diagnostic information is made available by the CPU for each analog channel associated with analog modules and Genius analog blocks. One byte of diagnostic memory is allocated to each analog I/O channel. Since each analog I/O channel uses two bytes of %AI and %AQ memory, the diagnostic memory is half the size of the data memory.

The analog diagnostic data contains both diagnostics and process data with the process data being the High Alarm and Low Alarm bits. The diagnostic data is referenced with the -[F]- and -[NF]- contacts. The process bits are referenced with the high alarm (-[HA]- and low alarm (-[LA]-) contacts. The memory allocation for analog diagnostic data is one byte per word of analog input and analog output allocated by programming software. When an analog fault contact is referenced in the application program, the CPU does an Inclusive OR on all the bits in the diagnostic byte except the process bits. The alarm contact is closed if any diagnostic bit is ON and OFF only if all bits are OFF.

PACSystems Handling of GBC Faults

Defaulting of input data associated with failed/lost GBCs

When a GBC is missing, mismatched, or otherwise failed, the CPU applies the Input Default setting for each device on that Genius bus when defaulting the input data. If the device is configured for HOLD LAST STATE, the data is left alone. If the device is configured for OFF, the input data is set to 0. If a redundant GBC is operational, the input data is not affected.

Application of default input and diagnostic data for lost redundant blocks

When a GBC reports that a redundant block is lost, the CPU updates the input data tables and input diagnostic tables with the default data during the very next input scan. The output diagnostic data tables are updated during the very next output scan.

Power-Up and Power-Down Sequences

Power-Up Sequence

System power-up consists of the following parts:

- Power-up self-test
- CPU memory validation
- System configuration
- Intelligent option module self-test completion
- Intelligent option module dual port interface tests
- I/O system initialization

Power-Up Self-Test

On system power-up, many modules in the system perform a power-up diagnostic self-test. The CPU module executes hardware checks and software validity checks. Intelligent option modules perform setup and verification of on-board microprocessors, software checksum verification, local hardware verification, and notification to the CPU of self-check completion. Any failed tests are queued for reporting to the CPU during the system configuration portion of the cycle.

If a low or failed battery indication is present, a fault is logged in the CPU fault table.

CPU Memory Validation

The next phase of system power-up is the validation of the CPU memory. First, the system verifies that the battery is not low and that battery-backed RAM areas are still valid. A known area of battery-backed application RAM is checked to determine if data was preserved. Next, if a ladder diagram program exists, a checksum is calculated across the _MAIN ladder block. If no ladder diagram program exists, a checksum is calculated across the smallest standalone C program.

When the system is sure that the application RAM is preserved, a known area of the bit cache area is checked to determine if the bit cache data was preserved. If this test passes, the Bit Cache memory is left containing its power-up values. (Non-retentive outputs are cleared on a transition from Stop to Run mode.) If the checksum is not valid or the retentive test on the application RAM fails, the bit cache memory is assumed to be in error and all areas are cleared. The CPU is now in a cleared state, the same as if a new CPU module were installed. All logic and configuration files must be stored from the programmer to the CPU.

System Configuration

After completing its self-test, the CPU performs the system configuration. It first clears all of the system diagnostic bits in the bit cache memory. This prevents faults that were present before power-down, but are no longer present, from accidentally remaining as faulted. Then it polls each module in the system for completion of the module's self-test.

The CPU reads information from each module, comparing it with the stored (downloaded) rack/slot configuration information. Any differences between actual configuration and the stored configuration are logged in the fault tables.

Intelligent Option Module Self-Test Completion

Intelligent option modules may take a longer time to complete their self-tests than the CPU due to the time required to test communications media or other interface devices. As an intelligent option module completes its initial self-tests, it tells the CPU the time required to complete the remainder of these self-tests. During this time, the CPU provides whatever additional information the module needs to complete its self-configuration, and the module continues self-tests and configuration. If the module does not report back in the time it specified, the CPU marks the module as faulted and makes an entry in one of the fault tables. When all self-tests are complete, the CPU obtains reports generated during the module's power-up self-test and places fault information (if any) in the fault tables.

Intelligent Option Module Dual Port Interface Tests

After completion of the intelligent option module self-test and results reporting, integrity tests are jointly performed on the dual-port interface used by the CPU and intelligent option module for communications. These tests validate that the two modules are able to pass information back and forth, as well as verify the interrupt and semaphore capabilities needed by the communications protocol. After dual port interface tests are complete, the communications messaging system is initialized.

I/O System Initialization

If the module is an input module, no further configuration is required. If the module is an output module, the module is commanded to go to its default state. The output modules default to all outputs off at power-up and in failure mode, unless configured otherwise.

A bus transmitter module is interrogated about what expansion racks are present in the system. Based on the bus transmitter module's response, the CPU adds those racks and their associated slots into the list of slots to be configured.

Finally, the I/O Scanner performs its initialization. The I/O Scanner initializes all the I/O controllers in the system by establishing the I/O connections to each I/O bus on the I/O controller and obtaining all I/O configuration data from that I/O controller. This configuration data is compared with the stored I/O configuration and any differences reported in the I/O fault table. The I/O Scanner then sends each I/O controller a list of the I/O modules to be configured on the I/O bus. After the I/O controllers have been initialized, the I/O Scanner replaces the factory default settings in all I/O modules with any application-specified settings.

Power-Down Sequence

System power-down occurs when the power supply detects that incoming AC power has dropped for more than 15ms.

Retention of Data Memory Across Power Failure

Because application RAM is battery-backed, the following types of data are preserved across a power cycle:

- Application program
- Fault tables and other diagnostic data
- Checksums on programs and blocks
- Override data
- Data in register (%R), local register (%L), and program register (%P) memory
- Data in analog memory (%AI and %AQ)
- State of discrete inputs (%I)
- State of retentive discrete outputs (%Q)
- State of retentive discrete internals (%M)

The following types of data are not preserved across a power cycle:

- State of discrete temporary memory (%T)
- %M and %Q memories used on non-retentive -()- coils
- State of discrete system internals (system bits, fault bits, reserved bits)

Chapter 6

Program Organization

This chapter provides information about the operation of application programs in a PACSystems CPU.

- Structure of the Application Program
- Controlling Program Execution
- Interrupt-Driven Blocks

Structure of a PACSystems Application Program

A PACSystems application consists of one block-structured application program. The application program contains all the logic needed to control the operations of the CPU and the modules in the system. Application programs are created using the programming software and transferred to the CPU. Programs are stored in the CPU's non-volatile memory.

During the CPU Sweep (described in chapter 5), the CPU reads input data from the modules in the system and stores the data in its configured input memory locations. The CPU then executes the entire application program once, using this fresh input data. Executing the application program creates new output data that is placed in the configured output memory locations.

After the application program completes its execution, the CPU writes the output data to modules in the system.

A block-structured program always includes a _MAIN block. Program execution begins with the _MAIN block. Counting the _MAIN block, the program can contain up to 512 blocks.

Blocks

A block is a named section of executable logic that can be downloaded to and run on the target controller. The logic in a block can include functions, function blocks and calls to other blocks.

Functions and Function Blocks

A function is a type of instruction that has no internal storage (instance data). Therefore, it produces the same result for the same set of input values every time it executes.

A function block defines data as a set of inputs and output parameters that can be used as software connections to other blocks and internal variables. It has an algorithm that runs every time the function block is executed. Because a function block has instance data, that is it can store values, it has a defined state.

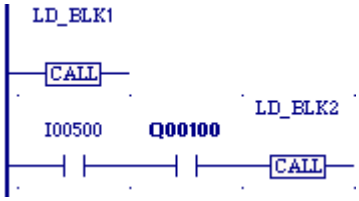
The following table describes the types of instructions that make up the PACSystems instruction set.

Instruction Type	*Instance Data*	*Examples*
Functions	None	BIT_SEQ, ADD, RANGE
Built-in function blocks	WORD array.	TMR, PID_IND, PID_ISA
Standard function blocks	Structure variable. (See “Instance Data Structures” on page 6-7.)	TP, TOF, TON

Note: A user defined function block (UDFB) is a block of logic that can be called in your program logic to create multiple instances of the block, allowing you to create a block of logic once and reuse it as if it was a standard function block instruction. For additional information, see pages 6-3 and 6-6.

How Blocks Are Called

A block executes when called from the program logic in the _MAIN block or another block. In this example, LD_BLK1 is always called. Conditional logic can be used to control calling a block. For LD_BLK2 to be called, input %I00500 and output %Q00100 must be ON. For details on using the Call function, refer to chapter 8 (LD programming), chapter 9 (FBD programming) or chapter 12 (ST programming).



Nested Calls

The CPU allows nested block calls as long as there is enough execution stack space to support the call. If there is not enough stack space to support a given block call, an “Application Stack Overflow” fault is logged. In these circumstances, the CPU cannot execute the block. Instead, it sets all of the block’s Boolean outputs to FALSE, and resumes execution at the point after the block call instruction.

Note: To halt the CPU when there is not enough stack space to execute a block, there are two choices. The best method is to add logic to detect the occurrence of any User Application Fault by testing the diagnostic bit %SA38, and then call SVC_REQ 13 to halt the CPU. An alternative method is to add logic that tests for a negative OK value coming out of the block and then call SVC_REQ 13 to halt the CPU.

A call depth of eight levels or more can be expected, except in rare cases where several of the called blocks have very large numbers of parameters. The actual call depth achieved depends on several factors, including the amount of data (non-Boolean) flow used in the blocks, the particular functions called by the blocks, and the number and types of parameters defined for the blocks. If blocks use less than the maximum amount of stack resources, more than eight nested calls may be possible. The call level nesting counts the _MAIN block as level 1.

Types of Blocks

PACSystems supports four types of blocks.

Block Type	*Local Data*	*Programming Languages*	*Size Limit*	*Parameters*
Block	Has its own local data	LD FBD ST	128 KB	0 inputs 1 output
Parameterized Block	Inherits local data from caller	LD FBD ST	128 KB	63 inputs 64 outputs
User Defined Function Block (UDFB)	Has its own local data	LD FBD ST	128 KB	63 inputs 64 outputs Unlimited internal member variables
External Block	Inherits local data from caller	C	user memory size limit (10 MB)	63 inputs 64 outputs

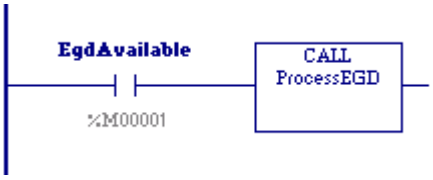
All PACSystems block types automatically provide an OK output parameter. The name used to reference the OK parameter within a block is Y0. Logic within the block can read and write the Y0 parameter. When a block is called, its Y0 parameter is automatically initialized to TRUE. This will result in a positive power flow out of the block call instruction when the block completes execution, unless Y0 is set to FALSE within the logic of the block.

For all block types, the maximum number of input parameters is one less than the maximum number of output parameters. This is because the EN input to the block call is not considered to be an input parameter to the block. It is used in LD language to determine whether or not to call the block, but is not passed into the block if the block is called.

Program Blocks

Any block can be a program block. The _MAIN block is automatically declared when you create a block-structured program. When you declare any other block, you must assign it a unique block name. A block is automatically configured with no input parameters and one output parameter (OK).

When a block-structured program is executed, the _MAIN block is automatically executed. Other blocks execute when called from the program logic in the _MAIN block, another block, or itself. In the following example, if %M00001 is ON, the block named ProcessEGD will be executed:



Program Blocks and Local Data

Program blocks support the use of %P global data. In addition, each block, except _MAIN, has its own %L local data. Blocks do not inherit %L local data from their callers.

Using Parameters With a Program Block

Every block is automatically defined to have one formal ‘power flow’ (or OK) output parameter, named Y0. Y0 is a BOOL parameter of LENGTH 1, passed by initial-value result. It indicates successful execution of the block. It can be read and written to by the logic within the block.

Parameterized Blocks

Any block except _MAIN can be a parameterized block. When you declare a parameterized block, you must assign it a unique block name. A parameterized block can be configured with up to 63 input and 64 output parameters.

A parameterized block executes when called from the program logic in the _MAIN block, another block, or itself. In the following example, if %I00001 is set, the parameterized block named LOAD_41 will be executed.



Parameterized Blocks and Local Data

Parameterized blocks support the use of %P global data. Parameterized blocks do not have their own %L data, but instead inherit the %L data of their calling blocks. Parameterized blocks also inherit the FST_EXE system reference and “time stamp” data that is used to update timer functions from their calling blocks. If %L references are used within a parameterized block and the block is called by _MAIN, %L references will be inherited from the %P references wherever encountered in the parameterized block (for example, %L0005 = %P0005).

Note: It is possible, by using Online Editing in the programming software to cause a parameterized block to use %L higher than allowed because of the way it inherits data. Using a word-for-word change to restore this reference to a valid address does not correct the block because the variable still exists in the variable list. Deleting the variable from the variable list does not cause an update to the CPU, so the parameterized block still sees the reference out of range fault. To correct this condition, you must remove the unused variables from the variable list after deleting them from the logic.

Using Parameters with a Parameterized Block

A parameterized block may be defined to have between 0 and 63 formal input parameters, and between 1 and 64 formal output parameters. A ‘power-flow out’ (or OK) parameter, named Y0, is automatically defined for every parameterized block. It is a BOOL parameter of LENGTH 1, and indicates the successful execution of the parameterized block. It can be read and written to by the parameterized block’s logic.

The following table lists the TYPEs, LENGTHs, and parameter passing mechanisms allowed for parameterized block parameters. (For definitions of the parameter passing types, see “Parameter Passing Mechanisms” on page 6-13.)

Type	*Length*	*Default Parameter Passing Mechanism*
BOOL	1 to 256	INPUTS: by value
		OUTPUTS: by value result; except Y0, which is by initial-value result
BYTE	1 to 1024	INPUTS: by reference
		OUTPUTS: by reference
INT, UINT, and WORD	1 to 512	INPUTS: by reference
		OUTPUTS: by reference
DINT, REAL, and DWORD	1 to 256	INPUTS: by reference
		OUTPUTS: by reference
LREAL	1 to 128	INPUTS: by reference
		OUTPUTS: by reference
function block*	1	INPUTS: by reference
		OUTPUTS: not allowed

* A maximum of 16 input parameters can be of type function block.

The PACSystems default parameter passing mechanisms correspond to the way that parameterized subroutine block (PSB) parameters are passed on 90-70 controllers. The parameter passing mechanisms of formal parameters cannot be changed from their default values.

Arguments, or “actual parameters” are passed into a parameterized block when a parameterized block call is executed. In general, arguments to formal parameters may come from any memory type, may be data flow, and may be constants (when the formal parameter’s LENGTH is 1). The following list contains the restrictions on arguments relative to this general rule:

- %S memory addresses cannot be used as arguments to any output parameter. This is because user logic is not allowed to write to %S memory.
- Indirect references used as arguments are resolved immediately before the parameterized block is called, and the corresponding direct reference is passed into the block. For example, where %R1 contains the value 10 and @R1 is used as an argument to a call, immediately before calling the block, @R1 is resolved to be %R10, and %R10 is passed in as the argument to the block. During execution of the block, the argument remains as %R10, regardless of whether the value in %R1 changes.

In general, formal parameters within a parameterized block may be used with any instruction, with any GE Fanuc function, or with any block call, as long as their TYPE and LENGTH are compatible with what the instruction, function, or block call requires. The following list contains the restrictions on formal parameters relative to this general rule:

- Formal parameters cannot be used on legacy transitional contacts or coils, or on FAULT, NOFLT, HIALM, or LOALM contacts. However, formal parameters can be used on IEC transitional contacts and coils.
- Formal BOOL input parameters cannot be used on coils or as output arguments to a GE Fanuc function or to a block call.
- Formal parameters cannot be used with the DO I/O function.
- Formal parameters cannot be used with indirect referencing.

User Defined Function Blocks

Users can define their own blocks, which have parameters and instance data, instead of being limited to the standard and built-in function blocks provided in the PACSystems instruction set. In many cases, the use of this feature results in a reduction in total program size.

A *member variable* is not passed into or out of a UDFB as a parameter. A member variable is used only within the logic of a function block.

Once defined, multiple instances of a UDFB can be created by calling it within the program logic. Each instance has its own unique copy of the function block's *instance data*, which consists of the function block's internal member variables and all of its input and output parameters except those that are passed by reference. When a UDFB is called on a given instance, the UDFB's logic operates on that instance's copy of the instance data. The values of the instance data persist from one execution of the UDFB to the next.

A UDFB cannot be triggered by an interrupt.

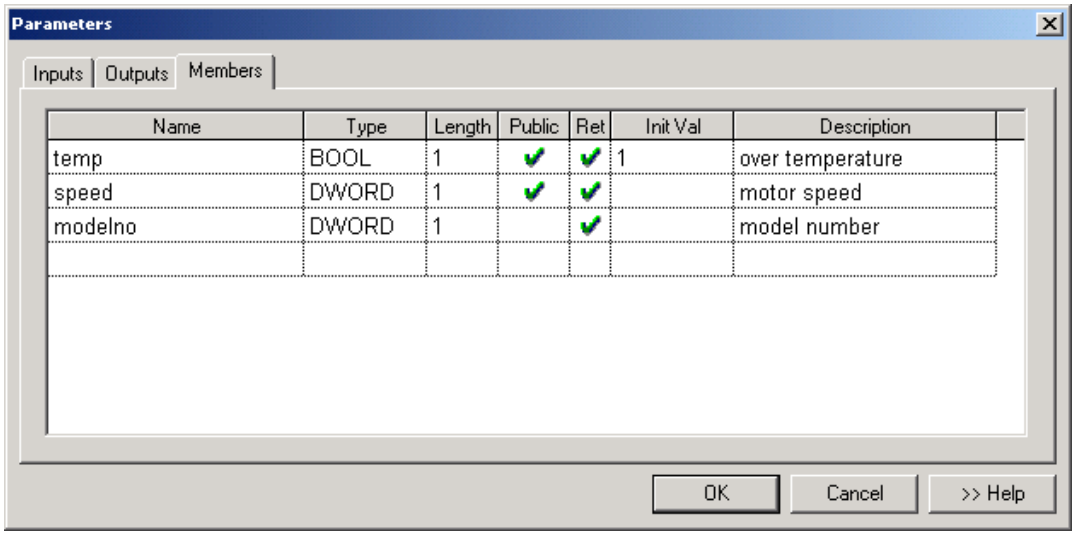
UDFB logic is created using FBD, LD or ST. UDFB logic can make calls to all the other types of PACSystems blocks (blocks, parameterized blocks, external blocks and other UDFBs). Blocks, parameterized blocks, and other UDFBs can make calls to UDFBs.

Unless otherwise stated, the PACSystems implementation of UDFBs meets the IEC 61131-3 requirements for user defined function blocks.

Defining a UDFB

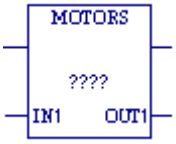
To create a UDFB in the programming software, create an LD, FBD or ST block in the Program Blocks folder. In the Properties for the block, select Function Block.

To define instance data for a UDFB, select Parameters in the block's properties. Input and output parameters are defined in the same way as for parameterized blocks. In the following example, three internal member variables are defined: ***temp, speed,*** and ***modelno.***

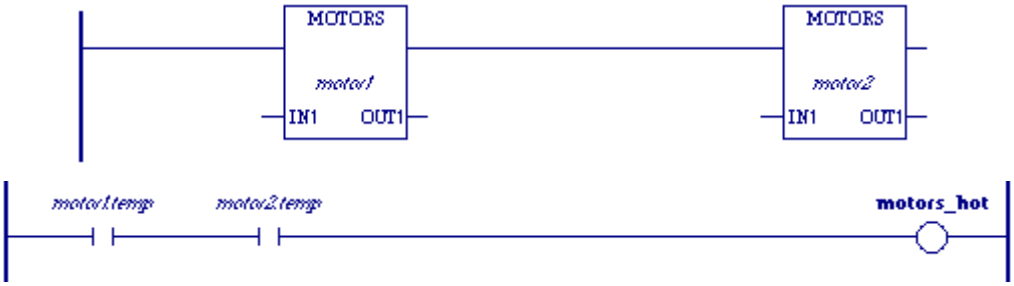


Creating UDFB Instances

You create an instance of a UDFB by calling it in your logic and assigning an instance name in the function properties.

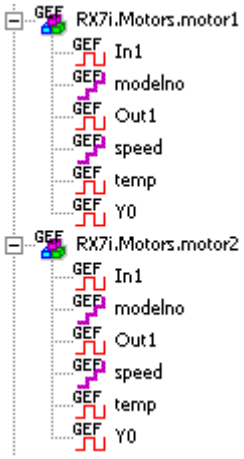


In the following LD example, the first rung creates two instances of the UDFB, Motors. The instance variables associated with the instances are motors.motor1 and motors.motor2. The second rung uses the two instances of the internal variable ***temp*** in logic.



Instance Data Structures

A variable with the format ***function_block_name.instance_name*** is automatically created for each instance of a UDFB. The instance data makes up a single composite variable that is of a structure type. The example to the right shows the variable structures associated with two instances of the UDFB named Motors. Each instance variable has elements corresponding to parameters ***In1, Out1***, and ***Y0***, and internal variables ***modelno, speed,*** and ***temp***.



Instances are created as symbolic variables, never as mapped variables. This ensures that instance data is only referenced by the instance name and not by a memory address, which means that no aliases can be created for the UDFB data elements. The indirect reference operator cannot be used on an instance variable because indirect references are not permitted on symbolic variables.

UDFBs and Scope

Unlike a parameterized subroutine, a UDFB has its own %L memory.

By default, internal variables of a UDFB have local scope, making them visible only to the logic inside the UDFB. They cannot be read or written by any external logic or by the hardware configuration. An internal variable can be made visible outside the UDFB by changing its scope to global. Logic outside the UDFB can read but cannot write to internal variables whose scope is global.

Note: If you give internal variables global scope, your application will not conform to IEC requirements.

Using Parameters with UDFBs

UDFBs support up to 63 inputs and up to 64 outputs.

Each UDFB has a predefined Boolean output parameter, Y0, which the CPU sets to true upon each invocation of the block. Y0 can be controlled by logic within the block and provides the output status of the block.

The following table lists the TYPEs, LENGTHs, and parameter passing mechanisms allowed for UDFB parameters. For additional information on parameter passing, see “Parameter Passing Mechanisms” on page 6-13.

Type	***Length***	***Parameter Passing Mechanism***	***Retentiveness of Instance Data for Parameters***
BOOL	1 to 256	INPUTS: by reference, value, or value result. (Default: value)	Not Applicable if passed by reference, since not stored in instance data. Can be retentive (default) or nonretentive for value or value result.
		OUTPUTS: by result; except Y0, which is by initial-value result	Retentive (default) or Nonretentive
BYTE	1 to 1024	INPUTS: by reference, value, or value result. (Default: value)	Retentive for value or value result.
		OUTPUTS: by result	Not applicable for reference
INT, UINT, and WORD	1 to 512	INPUTS: by reference, value, or value result. (Default: value)	Retentive for value or value result.
		OUTPUTS: by result	Not applicable for reference
DINT, REAL, and DWORD	1 to 256	INPUTS: by reference, value, or value result. (Default: value)	Retentive for value or value result.
		OUTPUTS: by result	Not applicable for reference
LREAL	1 to 128	INPUTS: by reference, value, or value result. (Default: value)	Retentive for value or value result.
		OUTPUTS: by result	Not applicable for reference
UDFB*	1	INPUTS: by reference	Not applicable since passed by reference
		OUTPUTS: not allowed	

* A maximum of 16 input parameters can be of type UDFB.

If an input parameter is passed by reference or by value result, it requires an argument. All other parameters of a UDFB are optional. That is, they do not have to be given arguments on each instance of the UDFB. If no argument is given for an optional parameter, the variable element associated with the parameter retains the value it previously had.

UDFB outputs cannot be passed as arguments to input parameters that are passed by reference or passed by value result. This restriction prevents modification of a UDFB output.

Using Internal Member Variables with UDFBs

A UDFB can have any number of internal member variables. Internal variables' values are not passed through the input and output parameters. An internal variable cannot have the same name as a parameter of the UDFB it is defined in.

An internal variable can be:

- Any basic type supported by PACSystems (BOOL, INT, UINT, DINT, REAL, LREAL, BYTE, WORD, and DWORD).
- A UDFB type. Such member variables are known as nested instances. For example, the function block "Motor" can have an internal variable of type "Valve," where Valve is a UDFB type. Note that defining a member variable as a UDFB type does not create an instance.

  A nested instance cannot be of the same type as the UDFB being defined because this would set up an infinitely recursive definition. Nor can any level of a nested instance be of the same type as the parent UDFB being defined. For example, the UDFB "Motor" cannot have an internal variable of type "Valve," if the Valve UDFB contains an internal variable of type "Motor."
- A one-dimensional array.

Internal variables of TYPE BOOL can be retentive (default) or nonretentive. All other TYPEs must be retentive.

Member variables corresponding to a UDFB's input parameters cannot be read or written outside of the UDFB. (This is more restrictive than the IEC 61131-3 requirements for user defined function blocks.) Member variables corresponding to the UDFB's output parameters can be read but not written outside the UDFB.

Internal member variables that have basic types may be given initial values. The same initial values apply to all instances of a UDFB. If an initial value isn't given, the internal member variable is set to zero when the application transitions to RUN mode for the first time.

An internal member variable that is a nested instance has initial values as specified by its UDFB type definition.

Initial values are not stored during a RUN mode store. They will not take effect until a STOP mode store is performed.

UDFB Logic

An instance of a BOOL parameter or internal variable can be forced ON or OFF, or used with transition-detecting instructions. The exception to this is that BOOL input parameters passed by reference cannot be forced or used with the Series 90-70 legacy transition-detecting instructions (POSCOIL, NEGCOIL, POSCON and NEGCON) because their values are not stored in instance data.

All input parameters to a UDFB, and their corresponding instance data elements, can be read by their UDFB's logic.

Input parameters that are passed by reference or passed by value result to a UDFB can be written to by their UDFB's logic. Input parameters passed by value ***cannot*** be written to by their UDFB logic. Note that the restriction on writing to input parameters passed by value does not apply to other types of blocks.

All UDFB output parameters can be both read and written to by their logic.

UDFB Operation with Other Blocks

A UDFB instance that is of global scope can be invoked by another UDFB's logic or any other block's logic.

A UDFB instance that is passed (by reference) as an argument to a UDFB can be invoked by the UDFB's logic.

A UDFB instance that is passed (by reference) as an argument to a parameterized block can be invoked by the parameterized block's logic.

The output parameters, and their corresponding instance data elements, of a UDFB instance that is passed as an argument can be read but not modified by the receiving block's logic. The input parameters of a UDFB instance that is passed as an argument cannot be read or modified by the receiving block's logic. The internal variables of a UDFB instance that is passed as an argument cannot be modified by the receiving block's logic. They can be read if their scope is global, but not if their scope is local.

External Blocks

External blocks are developed using external development tools as well as the C Programmer's Toolkit for PACSystems. Refer to the *C Programmer's Toolkit for PACSystems User's Manual,* GFK-2259 for detailed information regarding external blocks.

Any block except _MAIN can be an external block. When you declare an external block, you must assign it a unique block name. It can be configured with up to 63 input parameters and 64 output parameters.

An external block executes when called from the program logic in the _MAIN block or from the logic in another block, parameterized block, or UDFB. External blocks themselves cannot call any other block. In the following example, if %I00001 is set, the external block named EXT_11 is executed.



Note: Unlike other block types, external blocks cannot call other blocks.

External Blocks and Local Data

External blocks support the use of %P global data. External blocks do not have their own %L data, but instead inherit the %L data of their calling blocks. They also inherit the FST_EXE system reference and the "time stamp" data that is used to update timer function blocks from their calling blocks. If %L references are used within an external block and the block is called by _MAIN, %L references will be inherited from the %P references wherever encountered in the external block (for example, %L0005 = %P0005).

Initialization of C Variables

When an external block is stored to the CPU, a copy of the initial values for its global and static variables is saved. However, if static variables are declared without an initial value, the initial value is undefined and must be initialized by the C application. (Refer to “Global Variable Initialization” and “Static Variable” in the *C Programmer’s Toolkit for PACSystems,* GFK-2259). The saved initial values are used to re-initialize the block’s global and static variables whenever the CPU transitions from Stop to Run.

Using Parameters With an External Block

An external block may be defined to have between zero and 63 formal input parameters and between one and 64 formal output parameters. A ‘power-flow out’ (or OK) parameter, named Y0, is automatically defined for every external block. Y0 is a BOOL parameter of LENGTH 1, and indicates the successful execution of the block. It can be read and written to by the external block’s logic.

The following table gives the TYPEs, LENGTHs, and parameter passing mechanisms allowed for external block parameters.

Type	***Length***	***Default Parameter Passing Mechanism***
BOOL	1 to 256	INPUTS: by reference
		OUTPUTS: by reference; except Y0, which is by initial-value result
BYTE	1 to 1024	INPUTS: by reference
		OUTPUTS: by reference
INT, UINT, and WORD	1 to 512	INPUTS: by reference
		OUTPUTS: by reference
DINT, REAL, and DWORD	1 to 256	INPUTS: by reference
		OUTPUTS: by reference
LREAL	1 to 128	INPUTS: by reference
		OUTPUTS: by reference

The PACSystems default parameter passing mechanisms correspond to the way that external block parameters were passed on 90-70 controllers. At this time, the parameter passing mechanisms of formal parameters cannot be changed from their default values.

You must define a name for each formal input and output parameter. Parameter names are limited to between 1 and 3 characters so that they will show up on the Call instruction for the external block.

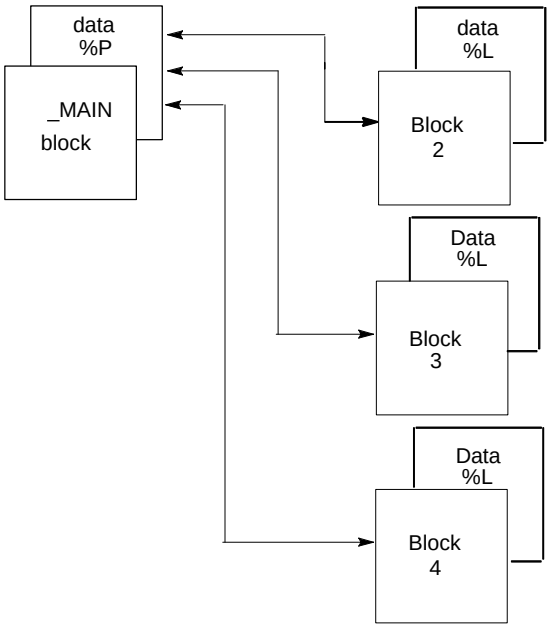
Arguments, or “actual parameters”, are passed into an external block when an external block call is executed.

Arguments may be any valid reference address including an indirect reference, may be flow, or may be a constant if the corresponding parameter’s LENGTH is 1.

Local Data

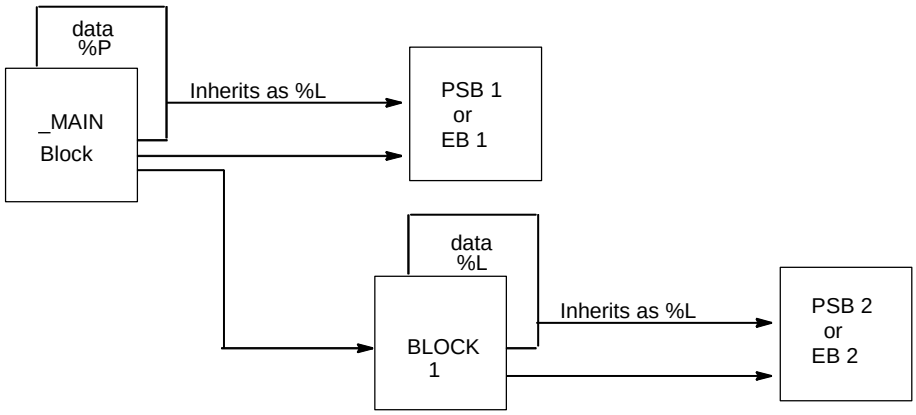
Each block or UDFB in a block-structured program has an associated local data block. _MAIN's data block memory is referenced by %P; all other data block memories are referenced by %L.

The size of the data block is dependent on the highest reference in its block for %L and in all blocks for %P.



All blocks within the program can use data associated with the _MAIN block (%P). Blocks and UDFBs can use their own %L data as well as the %P data that is available to all blocks. The _MAIN block cannot use %L.

External blocks and parameterized blocks can use the Local Data (%L) of their calling block as well as the %P data of the _MAIN block. If a parameterized block or external block is called by MAIN, all %L references in the parameterized block or external block will actually be references to corresponding %P references (for example, %L0005 = %P0005). In addition to inheriting the Local Data of their calling blocks, parameterized blocks and external blocks inherit the FST_EXE status of their calling blocks.



Parameter Passing Mechanisms

All blocks (except _MAIN) have at least one parameter and thus are affected by parameter passing mechanisms. A “parameter passing mechanism” describes the way that data is passed from an argument in a calling block to a parameter in the called block, and from the parameter in the called block back to the argument in the calling block.

PACSystems supports five different parameter passing mechanisms: pass by reference, pass by value, pass by value result, pass by result, and pass by initial-value result. A parameter is defined by its TYPE, LENGTH, and parameter passing mechanism.

When a parameter is **passed by reference**, the address of its argument is passed into the called block. All logic within the called block that reads or writes to the parameter is directly reading or writing to the actual argument.

When a parameter is **passed by value**, the value of its argument is copied into a local stack memory associated with the called block. All logic within the called block that reads or writes to the parameter is reading or writing to this stack memory. Thus no changes are ever made to the actual argument.

When a parameter is **passed by value result**, the value of its argument is copied into a local stack memory associated with the called block, and the address of its argument is saved. All logic within the called block that reads or writes to the parameter is reading or writing to this stack memory. When the called block completes its execution, the value in the stack memory is copied back to the actual argument’s address. Thus no changes are made to the actual argument while the called block is executing, but when it completes execution, the actual argument is updated.

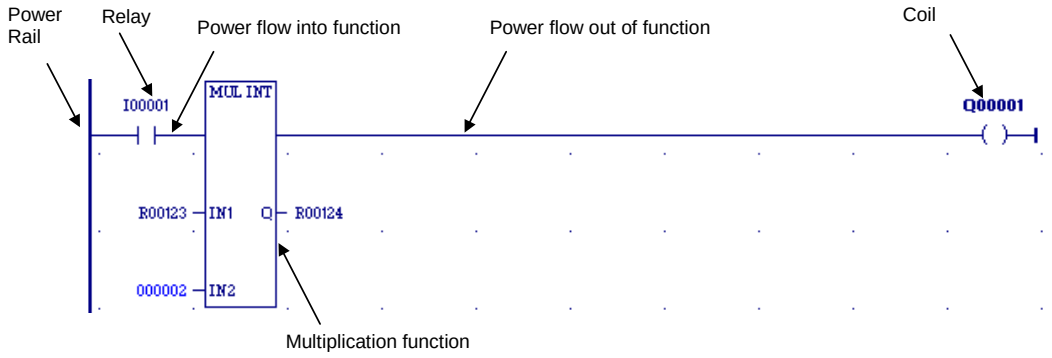
When a parameter is **passed by result**, space is allocated for its argument in a local stack memory associated with the called block, and the address of its argument is saved. All logic within the called block that reads or writes to the parameter is reading or writing to this stack memory. When the called block completes its execution, the value in the stack memory is copied back to the actual argument’s address. Thus no changes are made to the actual argument while the called block is executing, but when it completes execution, the actual argument is updated.

When a parameter is **passed by initial-value result**, an initial value of TRUE is copied into a local stack memory associated with the called block, and the address of its argument is saved. All logic within the called block that reads or writes to the parameter is reading or writing to this stack memory. When the called block completes its execution, the value in the stack memory is copied back to the actual argument’s address. Thus no changes are made to the actual argument while the called block is executing, but when it completes execution, the actual argument is updated. The OK output parameters of all blocks are passed by initial-value result.

Languages

Ladder Diagram (LD)

Logic written in Ladder Diagram language consists of a sequence of rungs that execute from top to bottom. The logic execution is thought of as “power flow”, which proceeds down along the left “rail” of the ladder, and from left to right along each rung in sequence.



The flow of logical power through each rung is controlled by a set of simple program instructions that work like mechanical relays and output coils. Whether or not a relay passes logical power flow along the rung depends on the content of a memory location with which the relay has been associated in the program. For instance, a relay might pass positive power flow if its associated memory location contains the value 1. The same relay passes negative power flow if the memory location contains the value 0.

Usually an instruction that receives negative power flow does not execute and propagates the negative power flow on to the next instruction in the rung. However, some instructions such as timers and counters execute even when they receive negative power flow, and may even pass positive power flow out. Once a rung completes execution, with either positive or negative power flow, power flows down along the left rail to the next rung.

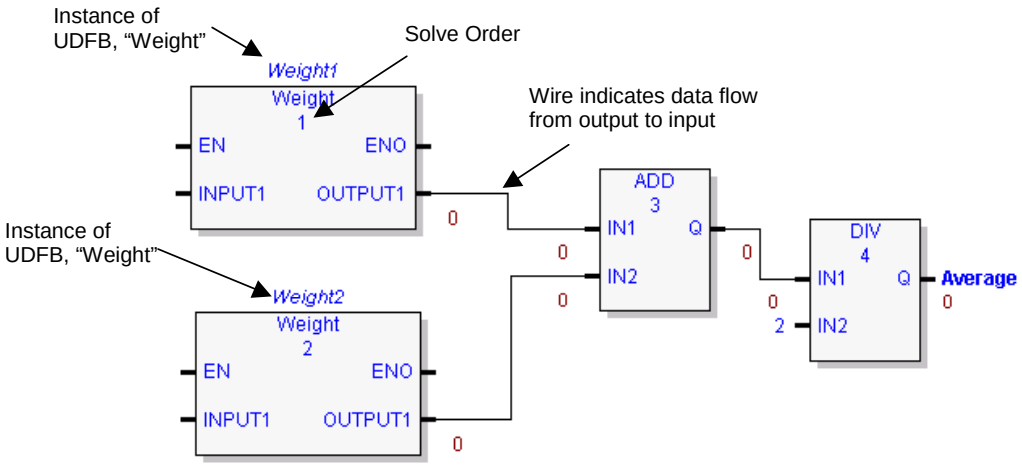
Within a rung, there are many complex functions that are part of the standard GE Fanuc function library and can be used for operations like moving data stored in memory, performing math operations, and controlling communications between the CPU and other devices in the system. Some program functions, such as the Jump function and Master Control Relay, can be used to control the execution of the program itself. Together, this large group of Ladder Diagram instructions and standard GE Fanuc library functions makes up the *instruction set* of the CPU.

Note: Series 90-70 PLCs use column-major execution: they execute a rung of LD logic by going from top to bottom, left to right, through each column in the rung. PACSystems and Series 90-30 CPUs use row-major execution: they execute an LD rung by tracing paths from left to right and top to bottom. For examples, refer to “Row- versus Column-major LD Logic Execution” in appendix C.

Function Block Diagram

Function Block Diagram (FBD) is an IEC 61131-3 graphical programming language that represents the behavior of functions, function blocks and programs as a set of interconnected graphical blocks.

FBD depicts a system in terms of the flow of signals between processing elements, in a manner very similar to signal flows depicted in electronic circuit diagrams. Instructions are shown with inputs entering from the left and outputs exiting on the right. A function block type name is always shown within the element and the name of the function block instance is shown above the element.



The order of execution of instructions in an FBD is determined by the following:

1. The display position of the instruction in the FBD editor
2. Whether the inputs to the FBD instruction are resolved.

To determine the order of execution of FBD instructions in the FBD editor, the FBD compiler performs the following steps:

1. The FBD compiler scans the instructions in the FBD editor, beginning from left to right, and top to bottom. When an instruction is encountered, the compiler attempts to resolve the instruction, that is, the inputs are known. If the inputs are known, the instruction is solved, and scanning continues for the next instruction.
2. If the current instruction cannot be resolved, that is, the inputs are not known, then the compiler scans for the previous instruction, using the wire connecting the output of the previous instruction to the input of the current instruction.
3. If the previous instruction can be resolved, the compiler calculates the output. The output of the previous instruction then becomes the input to the current instruction, the current instruction is resolved, and scanning continues for the next instruction.
4. If the previous instruction cannot be resolved, that is, the inputs are not known, then step 2 is repeated until an instruction is encountered, which can be resolved.

Structured Text

The Structured Text (ST) programming language is an IEC 1131-3 textual programming language. A structured text program consists of a series of statements, which are constructed from expressions and language keywords. A statement directs the PLC to perform a specified action. Statements provide variable assignments, conditional evaluations, iteration, and the ability to call other blocks. For details on ST statements, parameters, keywords, and operators supported by PACSystems, refer to chapter 12, “Structured Text.”

Blocks, parameterized blocks, and UDFBs can be programmed in ST. The _MAIN program block can also be programmed in ST.

A block programmed in ST can call blocks, parameterized blocks, and UDFBs.

Controlling Program Execution

There are many ways in which program execution can be controlled to meet the system’s timing requirements. The PACSystems CPU instruction set contains several powerful control functions that can be included in an application program to limit or change the way the CPU executes the program and scans I/O. For details on using these functions, refer to chapter 8.

The following is a partial list of the commonly used methods:

- The Jump (JUMPN) function can be used to cause program execution to move either forward or backward in the logic. When a JUMPN function is active, the coils in the part of the program that is skipped are left in their previous states (not executed with negative power flow, as they are with a Master Control Relay). Jumps cannot span blocks.
- The nested Master Control Relay (MCRN) function can be used to execute a portion of the program logic with negative power flow. Logic is executed in a forward direction and coils in that part of the program are executed with negative power flow. Master Control Relay functions can be nested to 255 levels deep.
- The Suspend I/O function can be used to stop both the input scan and output scan for one sweep. I/O can be updated, as necessary, during the logic execution through the use of DO I/O instructions.
- The Service Request function can be used to suspend or change the time allotted to the window portions of the sweep.
- Program logic can be structured so that blocks are called more or less frequently, depending on their importance and on timing constraints. The CALL function can be used to cause program execution to go to a specific block. Conditional logic placed before the Call function controls the circumstances under which the CPU executes the block logic. After the block execution is finished, program execution resumes at the point in the logic directly after the CALL instruction.

Interrupt-Driven Blocks

Three types of interrupts can be used to start a block's execution:

- **Timed Interrupts** are generated by the CPU based on a user-specified time interval with an initial delay (if specified) applied on Stop-to-Run transition of the CPU.
- **I/O Interrupts** are generated by I/O modules to indicate discrete input state changes (rising/falling edge), analog range limits (low/high alarms), and high speed signal counting events.
- **Module Interrupts** are generated by VME modules. A single interrupt is supported per module.

Caution

Interrupt-driven block execution can interrupt the execution of non-interrupt-driven logic. Unexpected results may occur if the interrupting logic and interrupted logic access the same data. If necessary, Service Request #17 or Service Request # 32 can be used to temporarily mask I/O and Timed Interrupt-driven logic from executing when shared data is being accessed.

Interrupt Handling

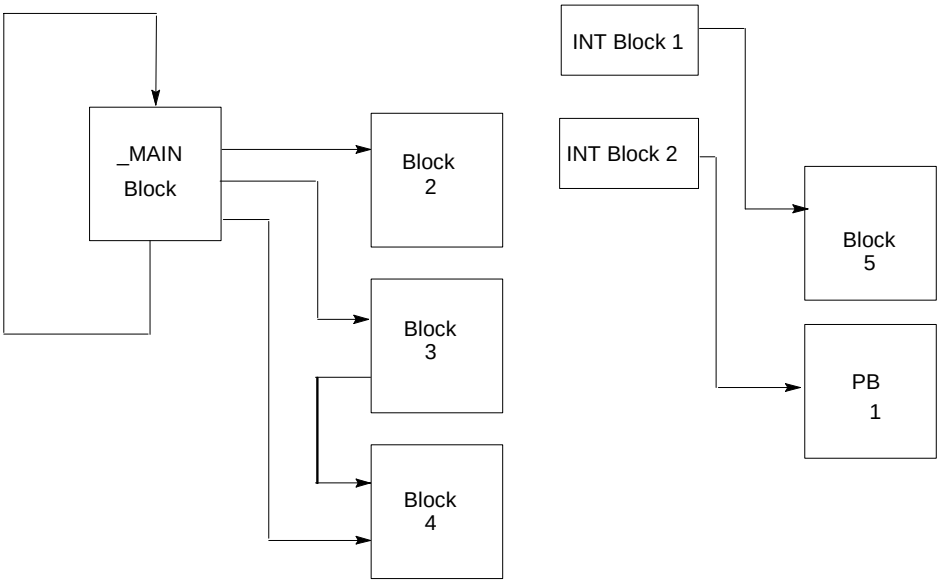
An I/O, Module, or Timed interrupt can be associated with any block except _MAIN, as long as the block has no parameters other than an OK output. After an interrupt has been associated with a block, that block executes each time the interrupt trigger occurs. A given block can have multiple timed, I/O, and module interrupt triggers associated with it. It is executed each time any one of its associated interrupts triggers. For details on how interrupt blocks are prioritized, refer to "Interrupt Block Scheduling" on page 6-19.

If a parameterized block or external block is triggered by an interrupt, it inherits %P data as its %L local data. For example, a %L00005 reference in the parameterized block or C block actually references %P00005.

Note: Timer function blocks do not accumulate time if used in a block that is executed as a result of an interrupt.

Blocks that are triggered by interrupts can make calls to other blocks. The application stack used during interrupt-driven execution is different from the stack used during normal block-structured program execution. In particular, the nested call limit is different from the limit described for calls from the _MAIN block. If a call results in insufficient stack space to complete the call, the CPU logs an "Application Stack Overflow" fault.

Note: GE Fanuc strongly recommends that interrupt-driven blocks not be called from the _MAIN block or other non-interrupt driven blocks because the interrupt and non-interrupt driven blocks could be reading and writing the same global memories at indeterminate times relative to each other. In the example below INT1, INT2, BLOCK5, and PB1 should not be called from _MAIN, BLOCK2, BLOCK3, or BLOCK4.



Timed Interrupts

A block can be configured to execute on a specified time interval with an initial delay (if specified) applied on a Stop-to-Run transition of the CPU.

To configure a timed interrupt block, specify the following parameters in the scheduling properties for the block:

Time Base	The smallest unit of time that you can specify for Interval and Delay. The time base can be 1.0 second, 0.10 second, or 0.01 second, or 0.001 second.
Interval	Specifies how frequently the block executes in multiples of the time base.
Delay	(Optional) Specifies an additional delay for the first execution of the block in multiples of the time base.

The first execution of a Timed Interrupt block will occur at
((delay * time base) + (interval * time base)) after the CPU is placed in Run mode.

I/O Interrupts

A block can be triggered by an interrupt input from certain hardware modules. For example, on the 32-Circuit 24 VDC Input Module (IC697MDL650), the first input can be configured to generate an interrupt on either the rising or falling edge of the input signal. If the interrupt is enabled in the module configuration, that input can serve as a trigger to cause the execution of a block.

To configure an I/O interrupt, specify a trigger in the scheduling properties for the block. The trigger must be a global variable in %I, %AI or %AQ memory, or an I/O variable. (An I/O variable is a form of symbolic variable that is mapped to a module I/O point in hardware configuration.)

Module Interrupts

A block can be triggered by an interrupt from a VME module if the VME Interrupt parameter is enabled in the module's hardware configuration. The PACSystems CPU supports one interrupt per module.

To configure a module interrupt, specify the module by rack/slot/interrupt ID as the Trigger in the scheduling properties for the block.

Interrupt Block Scheduling

You can select one of two types of interrupt block scheduling at the target level:

- **Normal block scheduling** allows you to associate a maximum of 64 I/O and Module Interrupts and 16 Timed Interrupts.. With normal block scheduling, all interrupt triggered blocks have equal priority. This is the default scheduling mode.
- **Preemptive block scheduling** allows you to associate a maximum of 32 interrupt triggers. With preemptive block scheduling, each trigger can be assigned a relative priority.

Normal Block Scheduling

Interrupt-driven logic has the highest priority of any user logic in the system. The execution of a block triggered from an interrupt preempts the execution of the normal CPU sweep activities. Execution of the normal CPU sweep activities is resumed after the interrupt-driven block execution completes.

If the CPU receives one or more interrupts while executing an interrupt block, it places the incoming interrupts into the queue while it finishes executing the current interrupt block. Timed interrupt driven blocks are queued ahead of I/O or Module driven blocks. I/O or Module interrupt driven blocks are queued in the order in which the interrupts are received. If an interrupt driven block is already in the queue, additional interrupts that occur for this block are ignored.

Preemptive Block Scheduling

Preemptive scheduling allows you to assign a priority to each interrupt trigger. The priority values range from 1 to 16, with 1 being the highest. A single block can have multiple interrupts with different priorities or the same priorities.

An incoming interrupt is handled according to its priority compared to that of the currently executing block as follows:

- If an incoming interrupt has a higher priority than the interrupt associated with the block that is currently executing, the currently executing block is stopped and put in the interrupt queue. The block associated with the incoming interrupt begins executing.
- If an incoming interrupt has the same priority as the interrupt trigger associated with the block that is currently executing, that block continues to execute and the incoming interrupt is placed in the queue.
- If an incoming interrupt has a lower priority than the interrupt associated with the block that is currently executing, the incoming interrupt is placed in the queue.

When the CPU completes the execution of an interrupt block, the block associated with the interrupt trigger that has the highest priority in the queue begins execution — or resumes execution if the block's execution was preempted by another interrupt block and was placed in the queue.

If multiple blocks in the queue have the same interrupt priority, their execution order is not deterministic.

Note: Certain functions, such as DOIO, BUS_RD, BUS_WRT, COMMREQ, SCAN_SET_IO, and some SVC_REQs may cause a block to yield to another queued block that has the same priority.

Chapter 7 Program Data

This chapter describes the types of data that can be used in an application program, and explains how that data is stored in the PACSystems CPU's memory.

- Variables
- Reference Memory
- User Reference Size and Default
- Genius Global Data
- Transitions and Overrides
- Retentiveness of Logic and Data
- Data Scope
- System Status References
- How Program Functions Handle Numerical Data
- Word-for-Word Changes
- Operands for Instructions

Variables

A variable is a named storage space for data values. It represents a memory location in the target PACSystems CPU.

A variable can be mapped to a reference address (for example, %R00001). If you do not map a variable to a specific reference address, it is considered a *symbolic variable*. The programming software handles the mapping for symbolic variables in a special portion of PACSystems user space memory.

The kinds of values a variable can store depends on its data type. For example, variables with a UINT data type store unsigned whole numbers with no fractional part. Data types are described in “How Program Functions Handle Numerical Data” on page 7-19.

In the programming software, all variables in a project are displayed in the Variables tab of the Navigator. You create, edit, and delete variables in the Variables tab. Some variables are also created automatically by certain components (such as TIMER variables when you add a Timer instruction to ladder logic). The data type and other properties of a variable, such as reference address are configured in the Inspector.

For more information about *system variables,* which are created when you create a target in the programming software, refer to page 7-14.

Mapped Variables

Mapped (manually located) variables are assigned a specific reference address. For details on the types of reference memory and their uses, refer to page 7-6.

Symbolic Variables

Symbolic variables are variables for which you do not specify a reference address (similar to a variable in a typical high-level language). Except as noted in this section, you can use these in the same ways that you use mapped variables.

In the programming software, a symbolic variable is displayed with a blank address. You can change a mapped variable to a symbolic variable by removing the reference address from the variable’s properties. Similarly, you can change a symbolic variable into a mapped variable by specifying a reference address for the variable in its properties.

The memory required to support symbolic variables counts against user space. The amount of space reserved for these variables is configured on the Memory tab in the CPU hardware configuration.

Restrictions on the Use of Symbolic Variables

- Symbolic variables cannot be used with indirect references (for example, @Name). For a description of indirect references, see page 7-6.
- Only global scope Symbolic variables can be used in EGD pages.
- A variable must be globally scoped and published (internal or external) to be used in a C block.
- Symbolic variables cannot be used in the COMMREQ status word.
- Use of symbolic variables is not supported on web pages.
- Symbolic Boolean variables are not allowed on non-BOOL parameters.
- Symbolic non-discrete variables cannot be used on Series 90-70 style Transitional contacts and coils. (Symbolic discrete variables are supported.)
- Overrides and Forces cannot be used on symbolic non-discrete variables. (Symbolic discrete variables are supported.)

I/O Variables

An I/O variable is a symbolic variable that is mapped to a terminal in the hardware configuration. A terminal can be one of the following: Physical discrete or analog I/O point on a PACSystems module or on a Genius device, a discrete or analog status returned from a PACSystems module, or Global Data. The use of I/O variables allows you to configure hardware modules without having to specify the reference addresses to use when scanning their inputs and outputs. Instead, you can directly associate variable names with a module's inputs and outputs.

As with symbolic variables, memory required to support I/O variables counts against user space. You can configure the space available for I/O variables in the Memory tab of the PACSystems CPU. For hardware configurations that were created with versions of Machine Edition 5.00 or earlier, the CPU will have to be updated to allocate space for I/O variables.

For a given module or Genius bus, you must use either I/O variables or manually located mapped variables: you cannot use both in combination. It is not necessary to map all points on a module. Points that are disconnected or unused can be skipped. When points are skipped, space is reserved in user memory for that point (that is, a 32-point discrete module will always use 32 bits of memory).

The hardware configuration (HWC) and logic become coupled in a PACSystems target on your computer as soon as you do one of the following: Enable I/O variables for a module or Genius bus (even if you don't create any I/O variables), use one or more symbolic variables in the Ethernet Global Data (EGD) component, or upload a coupled HWC and logic from a PACSystems PLC. The HWC and logic become coupled in a PACSystems PLC when coupled HWC and logic are downloaded to it.

Effects of coupled HWC and logic:

- Whether the HWC and logic are coupled in the PACSystems target on your computer or in the PACSystems PLC, you cannot download or upload the HWC and logic independently.
- When the HWC and logic are coupled in the PACSystems PLC, you cannot clear the HWC and the logic independently.
- As for any download, you cannot run mode store (RMS) the HWC and logic independently.
- The HWC must be completely equal for you to make word-for-word changes, launch the Online Test mode of Test Edit, or accept the edits of Test Edit.

I/O variables can be used any place that other symbolic variables are supported, such as in logic as parameters to built-in function blocks, user defined function blocks, parameterized function blocks, C blocks, bit-in-word references, and transitional contacts and coils.

Restrictions on the Use of I/O Variables

- Since I/O variables are a form of symbolic variable, the same restrictions that apply to other symbolic variables of the same data type and array bounds apply to I/O variables.
- Only a global variable can become an I/O variable. A local variable cannot become an I/O variable.
- You can map only a BOOL variable to a discrete terminal.
- You can map only an INT, UINT, or WORD variable to an analog terminal.
- You can map an entire array as an I/O variable if its number of elements is smaller than, or equal to, the number of terminals in the reference address node counting from and including the terminal where you enter the name of the array head or drop the array head. For example, if you have 32 analog terminals and you have a WORD array of 12 elements, you can map it to terminal 21 or any terminal before it (1 through 20).
- You can map a discrete array only to a terminal 8n+1, where n = 0, 1, 2, and so on. The "+1" is there because the terminals are numbered beginning with 1. If you map it to a terminal other than 8n+1, an error occurs upon validation.
- An I/O variable cannot be mapped to more than one location in hardware configuration.
- For the DO_IO function block, if an I/O variable is assigned to the ST parameter, then the same I/O variable must also be assigned to the END parameter, and the entire module is scanned.
- Some I/O modules do not currently support the use of I/O variables. For a list of modules that support I/O variables, please refer to the *Important Product Information* for Logic Developer – PLC programming software.

I/O Variable Format

When you map an I/O variable, the format used is %vdr.s.[z.]g.t:

v = I (input) or Q (output)

d = data type: X (BOOL) or W (WORD or analog). B (BYTE) and D (DWORD) are currently unsupported.

r = rack number

s = slot number

[z] = subslot number. This element and the period that follows it appear only if there is a subslot, for example, the SBA number of a Genius device. This value is set to 0 for a PACSystems RX7i Ethernet daughterboard.

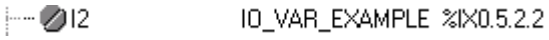
g = segment number or number of the reference address node. Set to 0 for the first reference address node on the Terminals tab, to 1 for the second reference node, and so on.

t = terminal number. This number is one-based, that is, the numbering begins at 1.

I/O Variable Examples



The I/O variable, Sample_IO_Variable is mapped to a non-discrete (W) output point (Q) on the module located in rack 0, slot 8. The variable is mapped to the first point in the first group of non-discrete output reference addresses.



The I/O variable, IO_VAR_EXAMPLE, is mapped to a discrete (X) input point (I) on the module located in rack 0, slot 5. The point is located in the module's third group of discrete input points and is point 2 in that group.

Reference Memory

The CPU stores program data in bit memory and word memory. Both types of memory are divided into different types with specific characteristics. By convention, each type is normally used for a specific type of data, as explained below. However, there is great flexibility in actual memory assignment.

Memory locations are indexed using alphanumeric identifiers called references. The reference's letter prefix identifies the memory area. The numerical value is the offset within that memory area, for example %AQ0056.

Word (Register) References

Type	Description
%AI	The prefix %AI represents an analog input register. An analog input register holds the value of one analog input or other non-discrete value.
%AQ	The prefix %AQ represents an analog output register. An analog output register holds the value of one analog output or other non-discrete value.
%R	Use the prefix %R to assign system register references that will store program data such as the results of calculations.
%W	Retentive Bulk Memory Area, which is referenced as %W (WORD memory).
%P	Use the prefix %P to assign program register references that will store program data with the _MAIN block. This data can be accessed from all program blocks. The size of the %P data block is based on the highest %P reference in all blocks. %P addresses are available only to the LD program they are used in, including C blocks called from LD blocks; they are not system-wide.

Note: All register references are retained across a power cycle to the CPU.

Indirect References

An indirect reference allows you to treat the contents of a variable assigned to an LD instruction operand as a pointer to other data, rather than as actual data. Indirect references are used only with word memory areas (%R, %W, %AI, %AQ, %P, and %L). An indirect reference in %W requires two %W locations as a DWORD indirect index value. For example, @%W0001 would use the %W2:W1 as a DWORD index into the %W memory range. The DWORD index is required because the %W size is greater than 65K.

Indirect references cannot be used with symbolic variables.

To assign an indirect reference, type the @ character followed by a valid reference address or variable name. For example, if %R00101 contains the value 1000, @R00101 instructs the CPU to use the data location of %R01000.

Indirect references can be useful when you want to perform the same operation to many word registers. Use of indirect references can also be used to avoid repetitious logic within the application program. They can be used in loop situations where each register is incremented by a constant or by a value specified until a maximum is reached.

Bit in Word References

Bit in word referencing allows you to specify individual bits in a word reference type as inputs and outputs of Boolean expressions, functions, and calls that accept bit parameters (such as parameterized blocks). This feature is restricted to word references in retentive memory. The bit number in the bit within word construct must be a constant.

You can use the programmer or an HMI to set an individual bit on or off within a word, or monitor a bit within a word. Also, C blocks can read, modify, and write a bit within a word.

Bit in Word references can be used in the following situations:

- In retentive 16-bit memory (AI, AQ, R, W, P, and L) and symbolics.
- On all contacts and coils ***except*** legacy transition contacts (POSCON/NEGCON) and transition coils (POSCOIL/NEGCOIL).
- On all functions and call parameters that accept single or unaligned bit parameters.

Functions that accept unaligned discrete references	***Parameters***
ARRAY MOVE (BIT)	SR and DS
ARRAY RANGE (BIT)	Q
MOVE (BIT)	IN and Q
SHFR (BIT)	IN, ST and Q

The use of Bit in Word references has the following restrictions:

- Bit in Word references cannot be used on legacy transition contacts (POSCON/NEGCON) and transition coils (POSCON/NEGCON).
- The bit number (index) must be a constant; it cannot be a variable.
- Bit addressing is not supported for a constant.
- Indirect references cannot be used to address bits in 16-bit memory.
- You cannot force a bit within 16-bit memory.

Examples:

%R2.X [0] addresses the first (least significant) bit of %R2

%R2.X [1] addresses the second bit of %R2. In the examples

In the examples [0] and [1] are the bit indexes. Valid bit indexes for the different variable types are:

BYTE variable	[0] through [7]
WORD, INT, or UINT variable	[0] through [15]
DWORD or DINT variable	[0] through [31]

Bit (Discrete) References

Type	Description
%I	Represents input references. %I references are located in the input status table, which stores the state of all inputs received from input modules during the last input scan. A reference address is assigned to discrete input modules using your programming software. Until a reference address is assigned, no data will be received from the module. %I memory is always retentive.
%Q	Represents physical output references. The coil check function checks for multiple uses of %Q references with relay coils or outputs on functions. You can select the level of coil checking desired (Single, Warn Multiple, or Multiple). %Q references are located in the output status table, which stores the state of the output references as last set by the application program. This output status table's values are sent to output modules at the end of the program scan. A reference address is assigned to discrete output modules using your programming software. Until a reference address is assigned, no data is sent to the module. A particular %Q reference may be either retentive or non-retentive.
%M	Represents internal references. The coil check function of your programming software checks for multiple uses of %M references with relay coils or outputs on functions. A particular %M reference may be either retentive or non-retentive.
%T	Represents temporary references. These references are never checked for multiple coil use and can, therefore, be used many times in the same program even when coil use checking is enabled—this is not a recommended practice because it makes subsequent trouble-shooting more difficult. %T may be used to prevent coil use conflicts while using the cut/paste and file write/include functions. Because this memory is intended for temporary use, it is cleared on Stop-to-Run transitions and cannot be used with retentive coils.
%S %SA %SB %SC	Represent system status references. These references are used to access special CPU data such as timers, scan information, and fault information. For example, the %SC0012 bit can be used to check the status of the CPU fault table. Once the bit is set on by an error, it will not be reset until after the sweep. %S, %SA, %SB, and %SC can be used on any contacts. ■ %SA, %SB, and %SC can be used on retentive coils -(M)-. **Note:** Although the programming software forces the logic to use retentive coils with %SA, %SB, and %SC references, most of these references are not preserved across battery-backed power cycles. %S can be used as word or bit-string input arguments to functions or function blocks. %SA, %SB, and %SC can be used as word or bit-string input or output arguments to functions and function blocks. For a description of the behavior of each bit, see "System Status References" on page 7-14.
%G	Represents global data references. These references are used to access data shared among several control systems.

Note: For details on retentiveness, refer to "Retentiveness of Logic and Data" on page 7-12.

User Reference Size and Default

Maximum user references and default reference sizes are listed in the table below.

Item	*Range*	*Default*
Reference Points		
%I reference	32768 bits	32768 bits
%Q reference	32768 bits	32768 bits
%M reference	32768 bits	32768 bits
%S total (S, SA, SB, SC)	512 bits (128 each)	512 bits (128 each)
%T reference	1024 bits	1024 bits
%G	7680 points	7680 points
Total Reference Points	107520	107520
Reference Words		
%AI reference	0—32640 words	64 words
%AQ reference	0—32640 words	64 words
%R, 1K word increments	0—32640 words	1024 words
%W	0—maximum available user RAM	0 words
Total Reference Words	0—maximum available user RAM	1152 words
%L (per block)	8192 words	8192 words
%P (per program)	8192 words	8192 words
Managed Memory		
Symbolic Discrete	0—83,886,080 (bits)	32768
Symbolic Non-Discrete	0—5,242,880 (words)	65536
I/O Discrete	0 through 83,886,080	0
I/O Non-Discrete	0 through 5,242,880.	0
Total Symbolic	0—42,088,704 bytes (This is the total memory available for the combined total of symbolic memory. This also includes other user memory use, program etc.)	143360

%G User References and CPU Memory Locations

The CPU contains one data space for all of the global data references (%G). The internal CPU memory for this data is 7680 bits long. For Series 90-70 systems, the programming software subdivides this range using %G, %GA, %GB, %GC, %GD, and %GE prefixes—allowing each of these prefixes to be used with bit offsets in the range 1–1280. For PACSystems, these ranges are converted to %G.

Genius Global Data

PACSystems supports the sharing of data among multiple control systems that share a common Genius I/O bus. This mechanism provides a means for the automatic and repeated transfer of %G, %I, %Q, %AI, %AQ, and %R data. No special application programming is required to use global data since it is integrated into the I/O scan. All GE Fanuc devices that have Genius I/O capability can send and receive global data from a PACSystems CPU.

Using I/O variables (page 7-3), you can directly associate variable names to a module's Genius global data that is scanned as part of an input/output scan.

Transitions and Overrides

The %I, %Q, %M, and %G user references, and symbolic variables of type BOOL, have associated transition and override bits. %T, %S, %SA, %SB, and %SC references have transition bits but not override bits. The CPU uses transition bits for counters, transitional contacts, and transitional coils. Note that counters do not use the same kind of transition bits as contacts and coils. Transition bits for counters are stored within the locating reference.

The transition bit for a reference tells whether the most recent value (ON, OFF) written to the reference is the same as the previous value of the reference. Therefore when a reference is written and its new value is the same as its previous value, its transition bit is turned OFF. When its new value is different from its previous value, its transition bit is turned ON. The transition bit for a reference is affected every time the reference is written to. The source of the write is immaterial; it can result from a coil execution, an executed function's output, the updating of reference memory after an input scan, etc.

When override bits are set, the associated references cannot be changed from the program or the input device; they can only be changed on command from the programmer. Overrides do not protect transition bits. If an attempted write occurs to an overridden memory location, the corresponding transition bit is cleared.

Retentiveness of Logic and Data

Data is defined as retentive if it is saved by the CPU when the CPU transitions from STOP mode to RUN mode.

The following items are retentive:

- program logic
- fault tables and diagnostics
- checksums for program logic
- overrides and output forces
- word data (%R, %W, %L, %P, %AI, %AQ)
- bit data (%I, %G, fault locating references, and reserved bits)
- %Q and %M variables that are configured as retentive (%T data is non-retentive and therefore not saved on STOP to RUN transitions.)
- symbolic variables that have a data type other than BOOL
- symbolic variables of BOOL type that are configured as retentive
- Retentive data is also preserved during battery-backed power-cycles of the CPU. Exceptions to this rule include the fault locating references and most of the %S, %SA, %SB, and %SC references. These references are initialized to zero at power-up regardless of the state of the battery. (See page 7-14 for a description of the behavior of each system status reference.)

When %Q or %M variables are configured as retentive, the contents are retained through power loss and Run-to-Stop-to-Run transitions.

Data Scope

Each of the user references has “scope”; that is, it may be available throughout the system, available to all programs, restricted to a single program, or restricted to local use within a block.

User Reference Type	***Range***	***Scope***
%I, %Q, %M, %T, %S, %SA, %SB, %SC, %G, %R, %W, %AI, %AQ, convenience references, fault locating references	Global	From any program, block, or host computer. Variables defined in these registers have system (global) scope by default. However, variables with local scope can also be assigned in these registers.
Symbolic variable	Global	From any program, block, or host computer. Symbolic variables have system (global) scope by default. However, symbolic variables with local scope can be created using the naming conventions for local variables.
I/O variable	Global	From any program, block, or host computer.
%P	Program	From any block, but not from other programs (also available to a host computer).
%L	Local	From within a block only (also available to a host computer).

In an LD block:

- %P should be used for program references that are shared with other blocks.
- %L are local references that can be used to restrict the use of register data to that block. These local references are not available to other parts of the program.
- %I, %Q, %M, %T, %S, %SA, %SB, %SC, %G, %R, %W, %AI, and %AQ references are available throughout the system.

System Status References

System status references in the CPU are assigned to %S, %SA, %SB, and %SC memory. The four timed contacts (time tick references) include #T_10MS, #T_100MS, #T_SEC, and #T_MIN. Examples of other system status references include #FST_SCN, #ALW_ON, and #ALW_OFF

Note: %S bits are read-only bits; do not write to these bits. You may, however, write to %SA, %SB, and %SC bits.

Listed below are available system status references that may be used in an application program. When entering logic, either the reference or the nickname can be used. Refer to chapter 15 for more detailed fault descriptions and information on correcting faults.

%S References

Reference	*Name*	*Definition*
%S0001	#FST_SCN	Current sweep is the first sweep in which the LD executed. Set the first time the user program is executed after a Stop/Run transition and cleared upon completion of its execution.
%S0002	#LST_SCN	Set when the CPU transitions to run mode and cleared when the CPU is performing its final sweep. The CPU clears this bit and then performs one more complete sweep before transitioning to Stop or Stop Faulted mode. If the number of last scans is configured to be 0, %S0002 will be cleared after the CPU is stopped and user logic will not see this bit cleared.
%S0003	#T_10MS	0.01 second timed contact.
%S0004	#T_100MS	0.1 second timed contact.
%S0005	#T_SEC	1.0 second timed contact.
%S0006	#T_MIN	1.0 minute timed contact.
%S0007	#ALW_ON	Always ON.
%S0008	#ALW_OFF	Always OFF.
%S0009	#SY_FULL	Set when the CPU fault table fills up (size configurable with a default of 16 entries). Cleared when an entry is removed from the CPU fault table and when the CPU fault table is cleared.
%S0010	#IO_FULL	Set when the I/O fault table fills up (size configurable with a default of 32 entries). Cleared when an entry is removed from the I/O fault table and when the I/O fault table is cleared.
%S0011	#OVR_PRE	Set when an override exists in %I, %Q, %M, or %G, or symbolic BOOL memory.
%S0012	#FRC_PRE	Set when force exists on a Genius point.
%S0013	#PRG_CHK	Set when background program check is active.
%S0014	#PLC_BAT	*CPE030, CRE030, CPE040, CRE040 and CPU320 only.* This contact is updated when a change in the battery status occurs. **Note:** This indication is not supported on these models and should be ignored: CPE010, CPE020, CRE020, and CPU310. To provide reliable backup, routine maintenance should include scheduled battery replacement. See “Battery Life Estimates” in Chapter 2 for specific CPU models.

Note: The #FST_EXE name is not associated with a %S address, it must be referenced by the name “#FST_EXE” only. This bit is set when transitioning from Stop to Run and indicates that the current sweep is the first time this block has been called.

%SA, %SB, and %SC References

Note: %SA, %SB, and %SC contacts are not set or reset until the input scan phase of the sweep following the occurrence of the fault or a clearing of the fault table(s). %SA, %SB, and %SC contacts can also be set or reset by user logic and CPU monitoring devices.

Reference	*Name*	*Definition*
%SA0001	#PB_SUM	Set when a checksum calculated on the application program does not match the reference checksum. If the fault was due to a temporary failure, the condition can be cleared by again storing the program to the CPU. If the fault was due to a hard RAM failure, then the CPU must be replaced. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0002	#OV_SWP	Set when the CPU detects that the previous sweep took longer than the time specified by the user. To clear this bit, clear the CPU fault table or power cycle the CPU. Only occurs if the CPU is in Constant Sweep mode.
%SA0003	#APL_FLT	Set when an application fault occurs. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0009	#CFG_MM	Set when a configuration mismatch fault is logged in the fault tables. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0008	#OVR_TMP	Set when the operating temperature of the CPU exceeds the normal operating temperature, 58ºC. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0010	#HRD_CPU	Set when the diagnostics detects a problem with the CPU hardware. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0011	#LOW_BAT	Not supported for the CPU battery and should be ignored. See page 7-18 for details.
%SA0012	#LOS_RCK	Set when an expansion rack stops communicating with the CPU. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0013	#LOS_IOC	Set when a Bus Controller stops communicating with the CPU. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0014	#LOS_IOM	Set when an I/O module stops communicating with the CPU. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0015	#LOS_SIO	Set when an option module stops communicating with the CPU. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0017	#ADD_RCK	Set when an expansion rack is added to the system. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SA0018	#ADD_IOC	Set when a Bus Controller is added to a rack. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0019	#ADD_IOM	Set when an I/O module is added to a rack. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0020	#ADD_SIO	Set when an intelligent option module is added to a rack. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0022	#IOC_FLT	Set when a Bus Controller reports a bus fault, a global memory fault, or an IOC hardware fault. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0023	#IOM_FLT	Set when an I/O module reports a circuit or module fault. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0027	#HRD_SIO	Set when a hardware failure is detected in an option module. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0029	#SFT_IOC	Set when there is a software failure in the I/O Controller. To clear this bit, clear the I/O fault table or power cycle the CPU.

Reference	Name	Definition
%SA0031	#SFT_SIO	Set when an option module detects an internal software error. To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0032	#SBUS_ER	Set when a bus error occurs on the VME bus backplane To clear this bit, clear the I/O fault table or power cycle the CPU.
%SA0081 – %SA0112		Set when a user-defined fault is logged in the CPU fault table. To clear these bits, clear the CPU fault table or power cycle the CPU. For more information, see discussion of Service Request 21 in chapter 10.
%SB0001	#WIND_ER	Set when there is not enough time to start the Programmer Window in Constant Sweep mode. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SB0009	#NO_PROG	Set when the CPU powers up with memory preserved, but no user program is present. Cleared when the CPU powers up with a program present or by clearing the CPU fault table.
%SB0010	#BAD_RAM	Set when the CPU detects corrupted RAM memory at power-up. Cleared when the CPU detects that RAM memory is valid at power-up or by clearing the CPU fault table.
%SB0011	#BAD_PWD	Set when a password access violation occurs. Cleared when the CPU fault table is cleared or when the CPU is power cycled.
%SB0012	#NUL_CFG	Set when an attempt is made to put the CPU in Run mode when there is no configuration data present. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SB0013	#SFT_CPU	Set when the CPU detects an error in the CPU operating system software. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SB0014	#STOR_ER	Set when an error occurs during a programmer store operation. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SB0016	#MAX_IOC	Set when more than 32 IOCs are configured for the system. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SB0017	#SBUS_FL	Set when the CPU fails to gain access to the bus. To clear this bit, clear the CPU fault table or power cycle the CPU.
%SC0009	#ANY_FLT	Set when any fault occurs that causes an entry to be placed in the CPU or I/O fault table. Cleared when both fault tables are cleared or when the CPU is power cycled.
%SC0010	#SY_FLT	Set when any fault occurs that causes an entry to be placed in the CPU fault table. Cleared when the CPU fault table is cleared or when the CPU is power cycled.
%SC0011	#IO_FLT	Set when any fault occurs that causes an entry to be placed in the I/O fault table. Cleared when the I/O fault table is cleared or when the CPU is power cycled.
%SC0012	#SY_PRES	Set as long as there is at least one entry in the CPU fault table. Cleared when the CPU fault table is cleared.
%SC0013	#IO_PRES	Set as long as there is at least one entry in the I/O fault table. Cleared when the I/O fault table is cleared.
%SC0014	#HRD_FLT	Set when a hardware fault occurs. Cleared when both fault tables are cleared or when the CPU is power cycled.
%SC0015	#SFT_FLT	Set when a software fault occurs. Cleared when both fault tables are cleared or when the CPU is power cycled.

Fault References

The fault references are discussed in chapter 15 of this manual but are presented here for your convenience.

System Fault References

System Fault Reference	Description
#ANY_FLT	Any new fault in either table since the last power-up or clearing of the fault tables
#SY_FLT	Any new system fault in the CPU fault table since the last power-up or clearing of the fault tables
#IO_FLT	Any new fault in the I/O fault table since the last power-up or clearing of the fault tables
#SY_PRES	Indicates that there is at lease one entry in the CPU fault table
#IO_PRES	Indicates that there is at least one entry in the I/O fault table
#HRD_FLT	Any hardware fault
#SFT_FLT	Any software fault

Configurable Fault References

Configurable Faults (Default Action	Description
#SBUS_ER (diagnostic)	System bus error. (The BSERR signal was generated on the VME system bus.)
#SFT_IOC (diagnostic)	Non-recoverable software error in a Genius Bus Controller.
#LOS_RCK (diagnostic)	Loss of rack (BRM failure, loss of power) or missing a configured rack.
#LOS_IOC (diagnostic)	Loss of Bus Controller missing a configured Bus Controller.
#LOS_IOM (diagnostic)	Loss of I/O module (does not respond) or missing a configured I/O module.
#LOS_SIO (diagnostic)	Loss of intelligent option module (does not respond) or missing a configured module.
#IOC_FLT (diagnostic)	Non-fatal bus or Bus Controller error—more than 10 bus errors in 10 seconds (error rate is configurable).
#CFG_MM (fatal)	Wrong module type detected during power-up, store of configuration, or Run mode. The CPU does not check the configuration parameters set up for individual modules such as Genius I/O blocks.

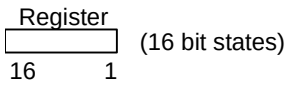
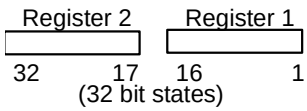
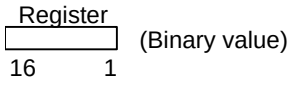
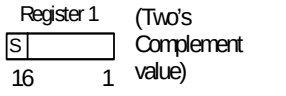
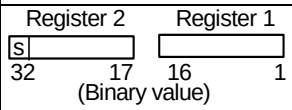
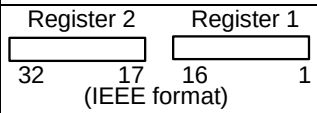
Non-Configurable Faults

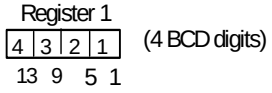
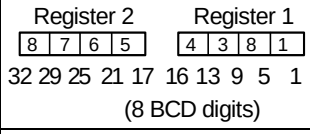
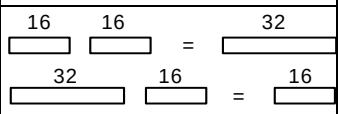
Non-Configurable Faults (Action)	Description
#SBUS_FL (fatal)	System bus failure. The CPU was not able to access the VME bus. BUSGRT-NMI error.
#HRD_CPU (fatal)	CPU hardware fault, such as failed memory device or failed serial port.
#HRD_SIO (diagnostic)	Non-fatal hardware fault on any module in the system.
#SFT_SIO (diagnostic)	Non-recoverable software error in a LAN interface module.
#PB_SUM (fatal)	Program or block checksum failure during power-up or in Run mode.
#LOW_BAT (diagnostic)	The low battery indication is not supported for the CPU battery. The CPU may set this bit when an I/O module or special-purpose module has reported a low battery. In this case, a fault will be reported in the I/O fault table. To clear this bit, clear the CPU fault table or power cycle the CPU.
#OV_SWP (diagnostic)	Constant sweep time exceeded.
#SY_FULL, IO_FULL (diagnostic)	CPU fault table full I/O fault table full
#IOM_FLT (diagnostic)	Point or channel on an I/O module—a partial failure of the module.
#APL_FLT (diagnostic)	Application fault.
#ADD_RCK (diagnostic)	New rack added, extra, or previously faulted rack has returned.
#ADD_IOC (diagnostic)	Extra I/O Bus Controller or reset of I/O Bus Controller.
#ADD_IOM (diagnostic)	Previously faulted I/O module is no longer faulted or extra I/O module.
#ADD_SIO (diagnostic)	New intelligent option module is added, extra, or reset.
#NO_PROG (information)	No application program is present at power-up. Should only occur the first time the CPU is powered up or if the battery-backed RAM containing the program fails.
#BAD_RAM (fatal)	Corrupted program memory at power-up. Program could not be read and/or did not pass checksum tests.
#WIND_ER (information)	Window completion error. Servicing of Programmer or Logic Window was skipped. Occurs in Constant Sweep mode.
#BAD_PWD (information)	Change of privilege level request to a protection level was denied; bad password.
#NUL_CFG (fatal)	No configuration present upon transition to Run mode. Running without a configuration is similar to suspending the I/O scans.
#SFT_CPU (fatal)	CPU software fault. A non-recoverable error has been detected in the CPU. May be caused by Watchdog Timer expiring.
#MAX_IOC (fatal)	The maximum number of bus controllers has been exceeded. The CPU supports 32 bus controllers.
#STOR_ER (fatal)	Download of data to CPU from the programmer failed; some data in CPU may be corrupted.

How Program Functions Handle Numerical Data

Regardless of where data is stored in memory – in one of the bit memories or one of the word memories – the application program can handle it as different data types.

Data Types

Type	Name	Description	Data Format
BOOL	Boolean	The smallest unit of memory. Has two states, 1 or 0. A BOOL array may have length N.	
BYTE	Byte	Has an 8-bit value. Has 256 values (0–255). A BYTE array may have length N.	
WORD	Word	Uses 16 consecutive bits of data memory. The valid range of word values is 0000 hex to FFFF hex.	
DWORD	Double Word	Has the same characteristics as a single word data type, except that it uses 32 consecutive bits in data memory instead of only 16 bits.	
UINT	Unsigned Integer	Uses 16-bit memory data locations. They have a valid range of 0 to +65535 (FFFF hex).	
INT	Signed Integer	Uses 16-bit memory data locations, and are represented in 2's complement notation. The valid range of an INT data type is –32768 to +32767.	 s=sign bit (0=positive, 1=negative)
DINT	Double Precision Integer	Stored in 32-bit data memory locations (two consecutive 16-bit memory locations). Always signed values (bit 32 is the sign bit). The valid range of a DINT data type is -2147483648 to +2147483647	 s=sign bit (0=positive, 1=negative)
REAL	Floating Point	Uses 32 consecutive bits (two consecutive 16-bit memory locations). The range of numbers that can be stored in this format is from ±1.401298E-45 to ±3.402823E+38. For the IEEE format, refer to "Floating Point Numbers" on page 7-20.	
LREAL	Double Precision Floating Point	Uses 64 consecutive bits (four consecutive 16-bit memory locations). The range of numbers that can be stored in this format is from ±2.2250738585072020E-308 to ±1.7976931348623157E+308. For the IEEE format, refer to "Floating Point Numbers" on page 7-20.	

Type	Name	Description	Data Format
BCD-4	Four-Digit BCD	Uses 16-bit data memory locations. Each binary coded decimal (BCD) digit uses four bits and can represent numbers between 0 and 9. This BCD coding of the 16 bits has a legal value range of 0 to 9999.	
BCD-8	Eight-Digit BCD	Uses two consecutive 16-bit data memory locations (32 consecutive bits). Each BCD digit uses 4 bits per digit to represent numbers from 0 to 9. The complete valid range of the 8-digit BCD data type is 0 to 99999999.	
MIXED	Mixed	Available only with the MUL and DIV functions. The MUL function takes two integer inputs and produces a double integer result. The DIV function takes a double integer dividend and an integer divisor to product an integer result.	
ASCII	ASCII	Eight-bit encoded characters. A single word reference is required to make two (packed) ASCII characters. The first character of the pair corresponds to the low byte of the reference word. The remaining 7 bits in each section are converted.	

Note: Using functions that are not explicitly bit-typed will affect transitions for all bits in the written byte/word/dword. For information about using floating point numbers, refer to “Floating Point Numbers” on page 7-20.

Floating Point Numbers

Floating point numbers are stored in one of two IEEE 754 standard formats that uses adjacent 16-bit words: 32-bit single precision or 64-bit double precision.

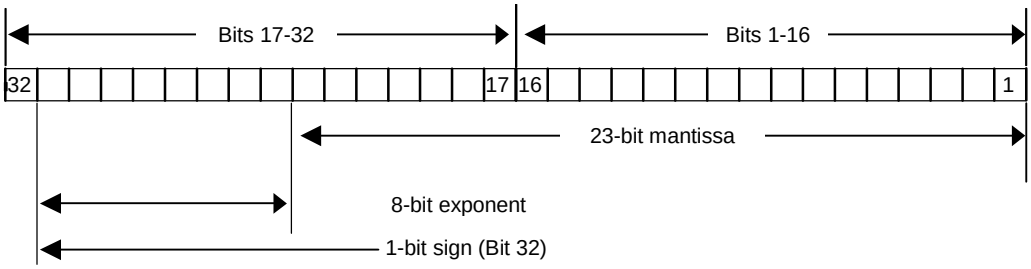
The REAL data type represents single precision floating point numbers. The LREAL data type represents double precision floating point numbers. REAL and LREAL variables are typically used to store data from analog I/O devices, calculated values, and constants.

Types of Floating Point Variables

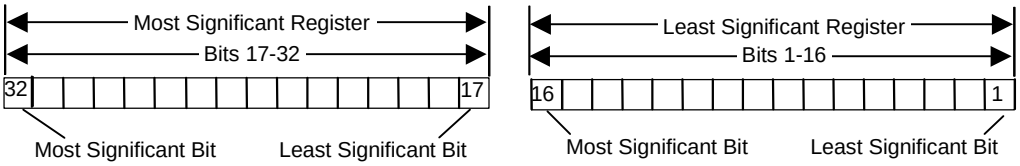
Data Type	Precision and Range
REAL	Limited to 6 or 7 significant digits, with a range of approximately $\pm 1.401298 \times 10^{-45}$ through $\pm 3.402823 \times 10^{38}$.
LREAL	Limited to 17 significant digits, with a range of approximately $\pm 2.225073858507202 \times 10^{-308}$ to $\pm 1.797693134862315 \times 10^{308}$.

Note: The programming software allows 32-bit and 64-bit arguments (DWORD, DINT, REAL, and LREAL) to be placed in discrete memories such as %I, %M, and %R in the PACSystems target. This is not allowed on Series 90-70 targets. (Note that any bit reference address that is passed to a non-bit parameter must be byte-aligned. This is the same as the Series 90-70 CPU.)

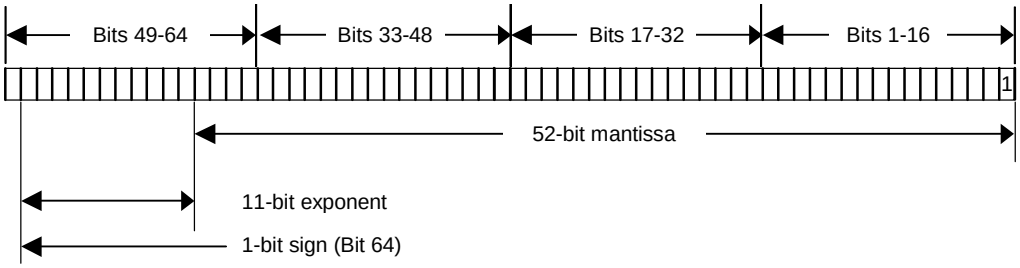
Internal Format of REAL Numbers



Register use by a single floating point number is diagrammed below. For example, if the floating point number occupies registers R5 and R6, R5 is the least significant register and R6 is the most significant register.



Internal Format of LREAL Numbers



Errors in Floating Point Numbers and Operations

Overflow occurs when a REAL or LREAL function generates a number outside the allowed range. When this occurs, the Enable Out output of the function is set Off, and the result is set to positive infinity (for a number greater than the upper limit) or negative infinity (for a number less than the lower limit). You can determine where this occurs by testing the sense of the Enable Out output.

IEEE 754 Infinity Representations

	REAL	***LREAL***
POS_INF (positive infinity)	= 7F800000h	= 7FF0000000000000h
NEG_INF (negative infinity)	= FF800000h	= 7FF0000000000001h

If the infinities produced by overflow are used as operands to other REAL or LREAL functions, they may cause an undefined result. This undefined result is referred to as an NaN (Not a Number). For example, the result of adding positive infinity to negative infinity is undefined. When the ADD_REAL function is invoked with positive infinity and negative infinity as its operands, it produces an NaN. If any operand of a function is a NaN, the result will be some NaN.

Note: For NaN, the Enable Out output is Off (not energized).

Binary representations of Infinity and NaN values have exponents that contain all 1s.

IEEE 754 Representations of NaN values:

REAL	***LREAL***
7F800001 through 7FFFFFFF	7FF8000000000001 through 7FFFFFFFFFFFFFFF
FF800001 through FFFFFFFF	FFF0000000000001 through FFFFFFFFFFFFFFFF

Note: For releases 5.0 and greater, the CPU may return slightly different values for NaN compared to previous releases. In some cases, the result is a special type of NaN displayed as #IND in Machine Edition. In these cases, for example, EXP(-infinity), power flow out of the function is identical to that in previous releases.

Word-for-Word Changes

Many changes to the program that do not modify the size of the program are considered word-for-word changes. Examples include changing the type of contact or coil, or changing a reference address used for an existing function block.

Symbolic Variables

Creating, deleting, or modifying a symbolic variable definition is not a word-for-word change.

The following are word-for-word changes:

- Switching between two symbolic variables
- Switching between an symbolic variable and a mapped variable
- Switching between a constant and a symbolic variable

Operands for Instructions

The operands for PACSystems instructions can be in the following forms:

- Constants
- Variables that are located in any of the PACSystems memory areas (%I, %Q, %M, %T, %G, %S, %SA, %SB, %SC, %R, %W, %L, %P, %AI, %AQ)
- Symbolic variables, including I/O variables
- Parameters of a Parameterized block or C block
- Power flow
- Data flow
- Computed references such as indirect references or bit-in-word references
- BOOL arrays

An operand's type and length must be compatible with that of the parameter it is being passed into. Otherwise, as few restrictions as possible are placed on operands. Many of the restrictions on older GE Fanuc PLCs have been removed from the PACSystems PLCs. PACSystems instructions and functions have the following operand restrictions:

- Constants cannot be used as operands to output parameters because output values cannot be written to constants.
- Variables located in %S memory cannot be used as operands to output parameters because %S memory is read-only.
- Variables located in %S, %SA, %SB, and %SC memories cannot be used as operands to numerical parameters such as INTs, DINTs, REALs, LREALs, etc.
- Data flow is prohibited on some input parameters of some functions. This occurs when the function, during the course of its execution, actually writes a value to the input parameter. Data flow is prohibited in these cases because data flow is stored in a temporary memory and any updated value assigned to it would be inaccessible to the user application.
- The arguments to EN, OK, and many other BOOLEAN input and output parameters are restricted to be power flow.
- Restrictions on using Parameterized block or External block parameters as operands to instructions or functions are documented in chapter 6.
- References in discrete memory (I, Q, M, and T) must be byte-aligned.

Note the following:

- Indirect references, which are available for all WORD-oriented memories (%R, %W, %P, %L, %AI, %AQ), can be used as arguments to instructions wherever located variables in the corresponding WORD-oriented memory are allowed. Note that indirect references are converted into their corresponding direct references immediately before they are passed into an instruction or function.
- Bit-in-word references are generally allowed on contact and coil instructions other than transition contacts and coils. They are also allowed as arguments to function parameters that accept single or unaligned bits.

BOOL arrays can be used as parameters to an instruction instead of variables of other data types. The array must be of sufficient length to replace the given data type. For example, instead of using a 16-bit INT variable, you could use a BOOL array of length 16 or more.

The following conditions must be met:

- The BOOL array must be byte-aligned, that is, the reference address of the first element of the BOOL array must be 8n + 1, where n = 0, 1, 2, 3, and so on. For example, %M00033 is byte-aligned, because 33 = (8 * 4) + 1.
- The parameter in question must support discrete memory reference addresses.
- The instruction in question must not have a Length parameter. (The Length parameter is displayed as ?? in the LD editor until a value is assigned.)
- The data type to be replaced with a BOOL array must be one of the following:

Data Type	***Minimum Length***
BYTE	8
INT, UINT, WORD	16
DINT, DWORD, REAL	32
REAL	64

- Excess bits are ignored. For example, if you use a BOOL array of length 12 instead of an 8-bit BYTE, the last four bits of the BOOL array are ignored.

Chapter 8 Ladder Diagram Programming

This chapter describes the programming instructions that can be used to create ladder logic programs for the PACSystems control system.

For an overview of the types of operands that can be used with instructions, refer to “Operands for Instructions” in chapter 7.

The ladder logic implementation of the PACSystems instruction set includes the following categories:

- Advanced Math 8-2
- Bit Operations 8-7
- Coils 8-25
- Contacts 8-31
- Control Functions 8-39
- Conversion Functions 8-58
- Counters 8-71
- Data Move Functions 8-75
- Data Table Functions 8-97
- Math Functions 8-115
- Program Flow Functions 8-125
- Relational Functions 8-135
- Timers 8-139

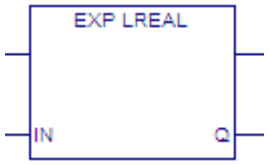
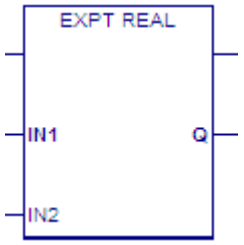
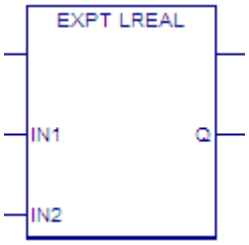
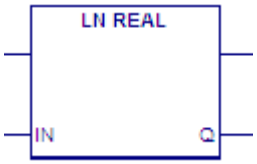
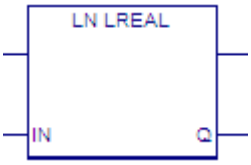
Advanced Math Functions

The Advanced Math functions perform logarithmic, exponential, square root, trigonometric, and inverse trigonometric operations.

Function	*Mnemonic*	*Description*
Exponential	EXP_REAL EXP_LREAL	Raises **e** to the value specified in IN ($\mathbf{e}^{IN}$). Calculates the inverse natural logarithm of the IN operand.
	EXPT_REAL EXPT_LREAL	Calculates IN1 to the IN2 power ($IN1^{IN2}$).
Inverse Trig	ACOS_REAL ACOS_LREAL	Calculates the inverse cosine of the IN operand and expresses the result in radians.
	ASIN_REAL ASIN_LREAL	Calculates the inverse sine of the IN operand and expresses the result in radians.
	ATAN_REAL ATAN_LREAL	Calculates the inverse tangent of the IN operand and expresses the result in radians.
Logarithmic	LN_REAL LN_LREAL	Calculates the natural logarithm of the operand IN.
	LOG_REAL LOG_LREAL	Calculates the base 10 logarithm of the operand IN.
Square Root	SQRT_DINT	Calculates the square root of the operand IN, a double-precision integer, and stores in Q the double-precision integer portion of the square root of the input IN.
	SQRT_INT	Calculates the square root of the operand IN, a single-precision integer, and stores in Q the single-precision integer portion of the square root of the input IN.
	SQRT_REAL SQRT_LREAL	Calculates the square root of the operand IN, a real number, and stores the real-number result in Q
Trig	COS_REAL COS_LREAL	Calculates the cosine of the operand IN, where IN is expressed in radians.
	SIN_REAL SIN_LREAL	Calculates the sine of the operand IN, where IN is expressed in radians.
	TAN_REAL TAN_LREAL	Calculates the tangent of the operand IN, where IN is expressed in radians.

Exponential/Logarithmic Functions

When an exponential or logarithmic function receives power flow, it performs the appropriate operation on the REAL or LREAL input value(s) and places the result in output Q.

The inverse natural log (EXP) function raises e to the power specified by IN.		
The Power of X (EXPT) function raises the value of input IN1 to the power specified by the value IN2.		
The Base 10 Logarithm (LOG) function calculates the base 10 logarithm of IN.		
The Natural Logarithm (LN) function calculates the logarithm of IN.		

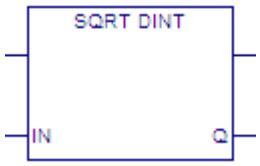
The power flow output is energized when the function is performed, unless overflow or one of the following invalid conditions occurs:

- IN < 0, for LOG or LN
- IN1 < 0, for EXPT
- IN is negative infinity, for EXP
- IN, IN1, or IN2 is a NaN (Not a Number)

Operands of the Exponential/Logarithmic Functions

Parameter	Description	Allowed Operands	Optional
IN or IN1	For EXP, LOG, and LN, IN contains the REAL or LREAL value to be operated on. The EXPT function has two inputs, IN1 and IN2. For EXPT, IN1 is the base value and IN2 is the exponent.	All except variables located in %S—%SC	No
IN2 (EXPT)	The REAL or LREAL exponent for EXPT.	All except variables located in %S—%SC	No
Q	Contains the REAL or LREAL logarithmic/exponential value of IN or of IN1 and IN2.	All except constants and variables located in %S—%SC	No

Square Root



Mnemonics:

SQRT_DINT
SQRT_INT
SQRT_REAL
SQRT_LREAL

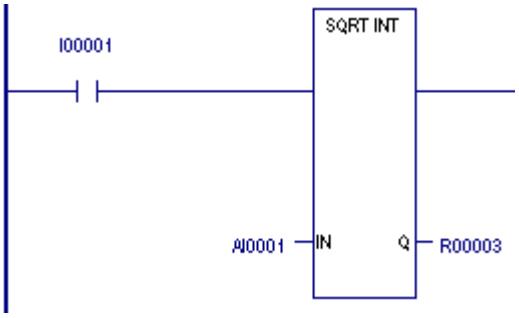
When the Square Root function receives power flow, it finds the square root of IN and stores the result in Q. The output Q must be the same data type as IN.

The power flow output is energized when the function is performed without overflow, unless one of these invalid REAL operations occurs:

- If IN < 0, Q is set to 0 and ENO is set FALSE.
- If IN is a NaN (Not a Number), Q will also be an NaN value and ENO will be set false.

Example

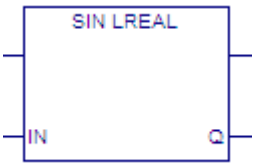
The square root of the integer number located at %AI0001 is placed into %R00003 when %I00001 is ON.



Operands for the Square Root Function

Parameter	Description	Allowed Operands	Optional
IN	The value to calculate the square root of. If IN < 0, the function does not pass power flow.	All except variables located in %S - %SC	No
Q	The calculated square root.	All except constants and variables located in %S - %SC	No

Trig Functions



Mnemonics:

SIN_REAL
SIN_LREAL
COS_REAL
COS_LREAL
TAN_REAL
TAN_LREAL

The SIN, COS, and TAN functions are used to find the trigonometric sine, cosine, and tangent, respectively, of an input whose units are radians. When one of these functions receives power flow, it computes the sine (or cosine or tangent) of IN and stores the result in output Q.

The SIN, COS, and TAN functions accept a broad range of input values, where $-2^{63} < IN < 2^{63}$, (2^{63} is approximately 9.22×10^{18}). Input values outside this range will produce incorrect results.

The power flow output is energized unless the following invalid condition occurs:

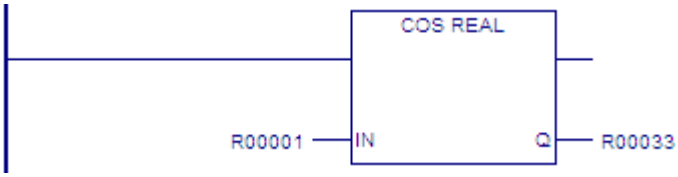
- IN or Q is a NaN (Not a Number)

Operands

Parameter	Description	Allowed Operands	Optional
IN	Number of radians. $-2^{63} < IN < 2^{63}$	All except variables located in %S—%SC	No
Q	Trigonometric value of IN (REAL or LREAL)	All except constants and variables located in %S—%SC	No

Example

The COS of the value in V_R00001 is placed in V_R00033.



Inverse Trig – ASIN, ACOS, and ATAN



Mnemonics:

ASIN_REAL
ASIN_LREAL
ACOS_REAL
ACOS_LREAL
ATAN_REAL
ATAN_LREAL

When an Inverse Sine (ASIN), Inverse Cosine (ACOS), or Inverse Tangent (ATAN) function receives power flow, it respectively computes the inverse sine, inverse cosine or inverse tangent of IN and stores the result in radians in output Q.

The ASIN and ACOS functions accept a narrow range of input values, where $-1 \le IN \le 1$. Given a valid value for the IN parameter, the ASIN function produces a result Q such that:

$$ASIN(IN) = -\frac{\pi}{2} \le Q \le \frac{\pi}{2}$$

The ACOS function produces a result Q such that:

$$ACOS(IN) = -0 \le Q \le \pi$$

The ATAN function accepts the broadest range of input values, where $-\infty \le IN \le +\infty$. Given a valid value for the IN parameter, the ATAN function produces a result Q such that:

$$ATAN(IN) = -\frac{\pi}{2} \le Q \le \frac{\pi}{2}$$

The power flow output is energized unless one of the following invalid conditions occurs:

- IN is outside the valid range for ASIN, ACOS, or ATAN
- IN is a NaN (Not a Number)

Operands of Inverse Trig Functions

Parameter	Description	Allowed Operands	Optional
IN	The REAL or LREAL value to process. ASIN and ACOS: $-1 \le IN \le 1$ ATAN: $-\infty \le IN \le +\infty$	All except variables located in %S - %SC	No
Q	Trigonometric value of IN. REAL or LREAL value expressed in radians. ASIN: $(-\pi/2) \le Q \le (\pi/2)$ ACOS: $0 \le Q \le \pi$ ATAN: $(-\pi/2) \le Q \le (\pi/2)$	All except constants and variables located in %S - %SC	No

Bit Operation Functions

The Bit Operation functions perform comparison, logical, and move operations on bit strings.

Function	*Mnemonics*	*Description*
Bit Position	BIT_POS_DWORD BIT_POS_WORD	Bit Position. Locates a bit set to 1 in a bit string.
Bit Sequencer	BIT_SEQ	Bit Sequencer. Sequences a string of bit values, starting at ST. Performs a bit sequence shift through an array of bits. The maximum length allowed is 256 words.
Bit Set, Clear	BIT_SET_DWORD BIT_SET_WORD	Bit Set. Sets a bit in a bit string to 1.
	BIT_CLR_DWORD BIT_CLR_WORD	Bit Clear. Clear a bit within a string by setting that bit to 0.
Bit Test	BIT_TEST_DWORD BIT_TEST_WORD	Bit Test. Tests a bit within a bit string to determine whether that bit is currently 1 or 0.
Logical AND	AND_DWORD AND_WORD	Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are both 1, places a 1 in the corresponding location in output string Q; otherwise, places a 0 in the corresponding location in Q.
Logical NOT	NOT_DWORD NOT_WORD	Logical invert. Sets the state of each bit in output bit string Q to the opposite state of the corresponding bit in bit string IN1.
Logical OR	OR_DWORD OR_WORD	Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are both 0, places a 0 in the corresponding location in output string Q; otherwise, places a 1 in the corresponding location in Q.
Logical XOR	XOR_DWORD XOR_WORD	Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are different, places a 1 in the corresponding location in the output bit string Q; when a pair of corresponding bits are the same, places a 0 in Q.
Masked Compare	MASK_COMP_DWORD MASK_COMP_WORD	Masked Compare. Compares the contents of two separate bit strings with the ability to mask selected bits.
Rotate Bits	ROL_DWORD ROL_WORD	Rotate Left. Rotates all the bits in a string a specified number of places to the left.
	ROR_DWORD ROR_WORD	Rotate Right. Rotates all the bits in a string a specified number of places to the right.
Shift Bits	SHIFTL_DWORD SHIFTL_WORD	Shift Left. Shifts all the bits in a word or string of words to the left by a specified number of places.
	SHIFTR_DWORD SHIFTR_WORD	Shift Right. Shifts all the bits in a word or string of words to the right by a specified number of places.

Data Lengths for the Bit Operation Functions

The Bit Operation functions operate on a single WORD or DWORD of data or up to 256 WORDs or DWORDs that occupy adjacent memory locations.

Bit Operation functions treat the WORD or DWORD data as a continuous string of bits, with bit 1 of the first WORD or DWORD being the Least Significant Bit (LSB). The last bit of the last WORD or DWORD is the Most Significant Bit (MSB). For example, if you specify three WORDs of data beginning at reference %R0100, they are treated as 48 contiguous bits.

%R0100	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1 ← bit 1 (LSB)
%R0101	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
%R0102	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33

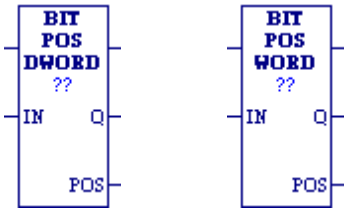
↑
(MSB)

Warning

Overlapping input and output reference address ranges in multiword functions is not recommended, as it can produce unexpected results.

Note that for all functions (Bit Test, Bit Set, Bit Clear, and Bit Position) that return a bit position indicator as an output parameter (POS), bit position numbering starts at 1, not 0, as shown in the diagram above.

Bit Position



The Bit Position function locates a bit set to 1 in a bit string.

Each scan that power is received, the function scans the bit string starting at IN. When the function stops scanning, either a bit equal to 1 has been found or the entire length of the string has been scanned.

POS is set to the position within the bit string of the first non-zero bit; POS is set to zero if no non-zero bit is found.

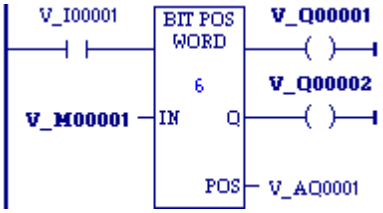
A string length of 1 to 256 WORDs or DWORDs can be selected. The function passes power flow to the right whenever it receives power.

Operands

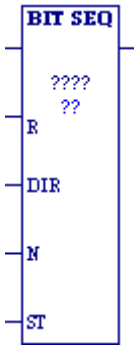
Parameter	Description	Allowed Operands	Optional
Length (??)	The number of WORDs or DWORDs in the bit string. 1 ≤ Length ≤ 256.	Constants	No
IN	The data to operate on	All. Constants may only be used when Length is 1.	No
Q	Energized if a bit set to 1 is found	Flow	Yes
POS	An unsigned integer giving the position of the first nonzero bit found, or zero if no non-zero bit is found	All except constants and variables located in %S - %SC	No

Examples

When V_I00001 is set, the bit string starting at V_M00001 is searched until a bit equal to 1 is found, or 6 words have been searched. Coil V_Q00001 is turned on. If a bit equal to 1 is found, its location within the bit string is written to V_AQ0001 and V_Q00002 is turned on. For example, if V_00001 is set, bit V_M00001 is 0, and bit V_M0002 is 1, the value written to V_AQ0001 is 2.



Bit Sequencer



The Bit Sequencer (BIT_SEQ) function performs a bit sequence shift through a series of contiguous bits.

The operation of BIT_SEQ depends on the value of the reset input (R), and both the current value and previous value of the enabling power flow input (EN):

R Current Execution	*EN Previous Execution*	*EN Current Execution*	*Bit Sequencer Execution*
ON	ON/OFF	ON/OFF	Bit sequencer resets
OFF	OFF	ON	Bit sequencer increments/decrements by 1
		OFF	Bit sequencer does not execute
	ON	ON/OFF	Bit sequencer does not execute

The reset input (R) overrides the enabling power flow (EN) and always resets the sequencer. When R is active, the current step number is set to the value of the optional N operand. If you did not specify N, the step number is set to 1. All bits in the bit sequencer, ST, are set to 0, except for the bit pointed to by the current step, which is set to 1.

When EN is active and R is not active, and the previous EN was OFF, the bit pointed to by the current step number is cleared. The current step number is incremented or decremented, based on the direction (DIR) operand. Then the bit pointed to by the new step number is set to 1.

- When the step number is being incremented and it goes outside the range of (1 ≤ step number ≤ Length), it is set back to 1.
- When the step number is being decremented and it goes outside the range of (1 ≤ step number ≤ Length), it is set to Length.

The parameter ST is optional. If it is not used, BIT_SEQ operates as described above, except that no bits are set or cleared. The function just cycles the current step number through its allowed range.

BIT_SEQ passes power to the right whenever it receives power.

Note: Before using the BIT_SEQUENCER function block, the current step number (Word 1 in the control block) must be set to an integer value between 1 and the length, as defined in the function block properties. Failure to properly initialize the step number in the BIT_SEQUENCER function block may result in the CPU going to STOP-HALT mode.

Asserting the Reset parameter (R), before using the BIT SEQUENCER function block assures that the current step number is set to a valid value.

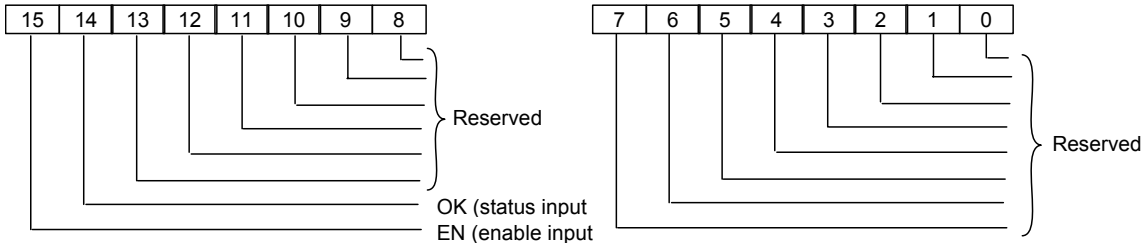
Memory Required for Bit Sequencer

Each bit sequencer uses a three word array of control block information. The control block can be a symbolic variable or it can be located in %R, %W, %L, or %P memory:

Word 1	current step number
Word 2	length of sequence (in bits)
Word 3	control word

Note: Do not write to the control block memory registers from other functions.

Word 3 (the control word) stores the state of the Boolean inputs and outputs of its associated function in the following format:



Notes:

- Bits 0 through 13 are not used.
- In the N operand, bits are entered as 1 through 16, not 0 through 15.

Operands for Bit Sequencer

Warning

Do not write to the Control Block memory with other instructions. Overlapping references may cause erratic operation of BIT_SEQ.

Parameter	*Description*	*Allowed Operands*	*Optional*
Address (????)	Beginning address of the Control Block, which is a three-word array: Word 1: current step number Word 2: length of sequence in bits Word 3: control word, which tracks the status of the last enabling power flow and the status of the power flow to the right.	Symbolic variables, variables located in %R, %W, %P, or %L	No
Length (??)	The number of bits in the bit sequencer, ST, that BIT_SEQ will step through. 1 ≤ Length ≤ 256.	Constants	No
R	When R is energized, the step number of BIT_SEQ is set to the value in N (default = 1), and the bit sequencer, ST, is filled with zeros, except for the current step number bit.	Flow	No
DIR	(Direction) When DIR is energized, the step number of BIT_SEQ is incremented prior to the shift. Otherwise, it is decremented.	Flow	No

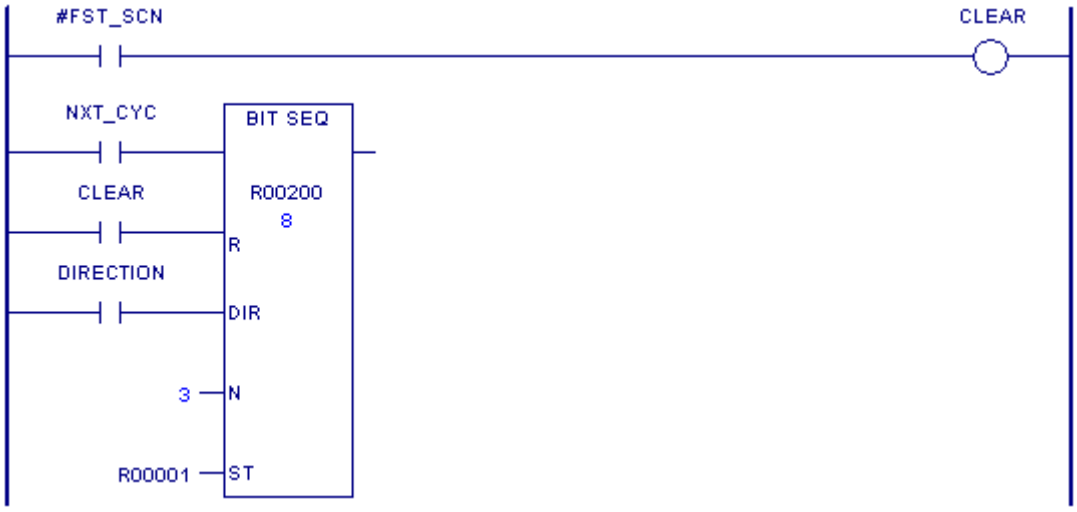
Parameter	Description	Allowed Operands	Optional
N	The value that the step number is set to when R is energized. Default value is 1. 1 ≤ N ≤ Length. If N < 1, the step number will be reset to 1 when R is energized. If N > Length, the step number will be reset to Length. Must be an integer variable or constant.	All except variables located in %S - %SC	Yes
ST	Contains the first word of the bit sequencer. If ST is not used, the Bit Sequencer function operates as described above, except that no bits are set or cleared. The function just cycles the current step number (in word 1 of the control block) through its allowed range. If ST is in %M memory and the Length is 3, the bit sequencer occupies 3 bits; the other 5 bits of the byte are not used. If ST is in %R memory, and the Length is 17, the bit sequencer uses 4 bytes, all of %R1 and %R2.	All except constants, flow, and variables located in %S	Yes

Example

In the following example, a #FST_SCN system variable is used to set CLEAR to ON for one scan. This sets the step number in Word 1 of the Bit Sequencer's control block to an initial value of 3.

The Bit Sequencer operates on register memory %R00001. Its control block is stored in registers %R0010, %R0011, and %R0012. When CLEAR is active, the sequencer is reset and the current step is set to step number 3, as specified in N. The third bit of %R0001 is set to one and the other seven bits are set to zero.

When NXT_CYC is active and CLEAR is not active, the bit for step number 3 is cleared and the bit for step number 2 or 4 (depending on whether DIRECTION is energized) is set.



Bit Set, Clear





Mnemonics

BIT_SET_DWORD
BIT_SET_WORD
BIT_CLR_DWORD
BIT_CLR_WORD

The Bit Set (BIT_SET_DWORD and BIT_SET_WORD) function sets a bit in a bit string to 1. The Bit Clear (BIT_CLR_DWORD and BIT_CLR_WORD) function clears a bit in a string by setting the bit to 0.

Each scan that power is received; the function sets or clears the specified bit. If a variable rather than a constant is used to specify the bit number, the same function can set or clear different bits on successive scans. Only one bit is set or cleared, and the transition information for that bit is updated. The transition status of all the other bits in the bit string is not affected.

The function passes power flow to the right, unless the value for BIT is outside the specified range.

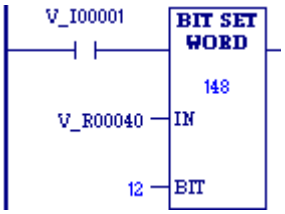
Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	The number of WORDs or DWORDs in the bit string. 1 ≤ Length ≤ 256.	Constants	
IN	The first WORD or DWORD of the data to process	All except constants, flow, and variables located in %S	
BIT	The number of the bit to set or clear in IN. 1 ≤ BIT ≤ (16 * Length) for WORD. 1 ≤ BIT ≤ (32 * length) for DWORD	All except variables located in %S - %SC	

Examples

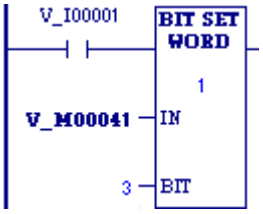
Example 1

Whenever input V_I0001 is set, bit 12 of the string beginning at reference %R00040 (as specified by variable V_R0040) is set to 1.

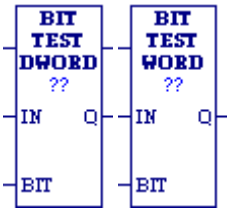


Example 2

Whenever V_I00001 is set, %M00043, the third bit of the string beginning at %M00041, is set to 1. Note that neither the status nor the transition value of any of the other bits in the same byte as %M00043 (e.g., %M00041, %M00042, %M00044, etc.) is affected by the BIT_SET function.



Bit Test



When the Bit Test function receives power flow, it tests a bit within a bit string to determine whether that bit is currently 1 or 0. The result of the test is placed in output Q.

Each scan that power is received, the Bit Test function sets its output Q to the same state as the specified bit. If a register rather than a constant is used to specify the bit number, the same function can test different bits on successive sweeps. If the value of BIT is outside the range (1 ≤ BIT ≤ (16 * length) for a WORD and 1 ≤ BIT ≤ (32 * length) for a DWORD), then Q is set OFF.

You can specify a string length of 1 to 256 WORDs or DWORDs.

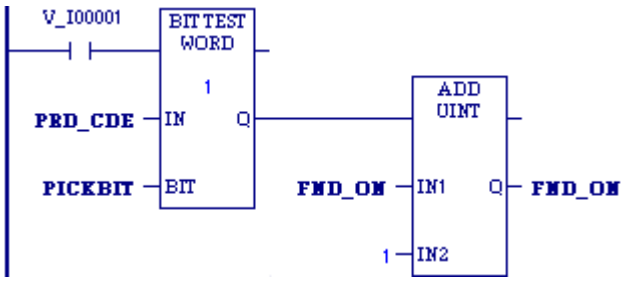
Note: When using the Bit Test function, the bits are numbered 1 through 16 for a WORD, *not* 0 through 15. They are numbered 1 through 32 for a DWORD.

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	The number of WORDs or DWORDs in the data string to test. 1 ≤ Length ≤ 256.	Constant	No
IN	The first WORD or DWORD in the data to test	All	No
BIT	The number of the bit to test in IN. 1 ≤ BIT ≤ (16*Length).	All except variables located in %S - %SC	No
Q	The state of the specific bit tested; Q is energized if the bit tested is a 1.	Flow	No

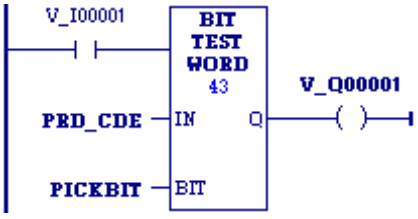
Example 1

When input V_I0001 is set, the bit at the location contained in reference PICKBIT is tested. The bit is part of string PRD_CDE. If it is 1, output Q passes power flow to the ADD function, causing 1 to be added to the current value of the ADD function input IN1.



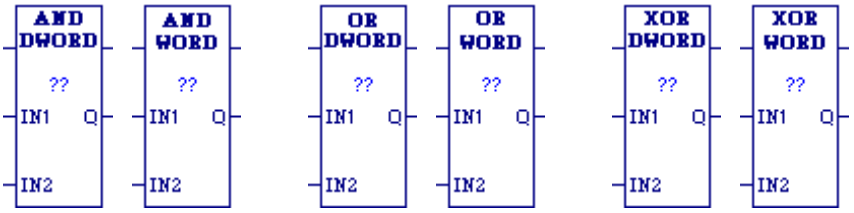
Example 2

When input V_I0001 is set, the bit at the location contained in reference PICKBIT is tested. The bit is part of string PRD_CDE. If it 1, output Q passes power flow and the coil V_Q0001 is turned on.



is

Logical AND, Logical OR, and Logical XOR



Each scan that power is received, the Logical function examines each bit in bit string IN1 and the corresponding bit in bit string IN2, beginning with the least significant bit in each. You can specify a string length of 1 to 256 WORDs or DWORDs. The IN1 and IN2 bit strings specified may overlap.

Logical AND

If both bits examined by the Logical AND function are 1, AND places a 1 in the corresponding location in output string Q. If either bit is 0 or both bits are 0, AND places a 0 in string Q in that location.

AND passes power flow to the right whenever it receives power.

Tip: You can use the Logical AND function to build masks or screens, where only certain bits are passed (the bits opposite a 1 in the mask), and all other bits are set to 0.

Logical OR

If either bit examined by the Logical OR function is 1, OR places a 1 in the corresponding location in output string Q. If both bits are 0, Logical OR places a 0 in string Q in that location. The function passes power flow to the right whenever it receives power.

Tips:

- You can use the Logical OR function to combine strings or to control many outputs with one simple logical structure. The Logical OR function is the equivalent of two relay contacts in parallel multiplied by the number of bits in the string.
- You can use the Logical OR function to drive indicator lamps directly from input states or to superimpose blinking conditions on status lights.

Logical XOR

When the Exclusive OR (XOR) function receives power flow, it compares each bit in bit string IN1 with the corresponding bit in string IN2. If the bits are different, a 1 is placed in the corresponding position in the output bit string.

For each pair of bits examined, if only one bit is 1, then XOR places a 1 in the corresponding location in bit string Q. XOR passes power flow to the right whenever it receives power.

Tips for Logical XOR

- If string IN2 and output string Q begin at the same reference, a 1 placed in string IN1 will cause the corresponding bit in string IN2 to alternate between 0 and 1, changing state with each scan as long as power is received.
- You can program longer cycles by pulsing the power flow to the function at twice the desired rate of flashing. The power flow pulse should be one scan long (one-shot type coil or self resetting timer).
- You can use XOR to quickly compare two bit strings, or to blink a group of bits at the rate of one ON state per two scans.
- XOR is useful for transparency masks.

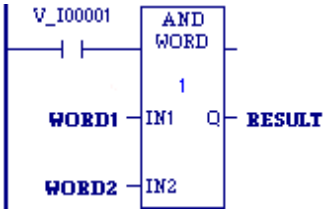
Operands for Logical AND, OR, and XOR

Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	The number of words in the bit string on which to perform the logical operation. 1 ≤ Length ≤ 256.	Constant	No
IN1	The first WORD or DWORD of the first string operate on.	All	No
IN2 (Must be the same data type as IN1.)	The first WORD or DWORD of the second string to operate on.	All	No
Q (Must be the same data type as IN1.)	The first WORD or DWORD of the operation's result.	All except constants and variables located in %S memory	No

Examples

Logical AND

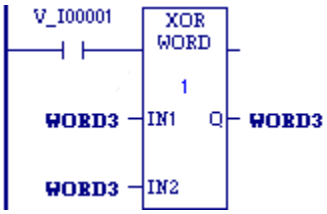
When input v_I0001 is set, the 16-bit strings represented by variables WORD1 and WORD2 are examined. The logical AND places the results in output string RESULT.



WORD1	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0
WORD2	1	1	0	1	1	1	0	0	0	0	0	0	1	1	1	1
RESULT	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0

Logical XOR

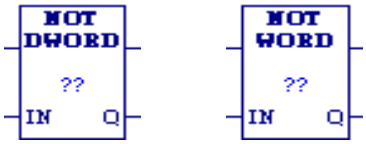
Whenever V_I0001 is set, the bit string represented by the variable WORD3 is cleared (set to all zeros).



I1 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0
I2 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0

Q (WORD3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Logical NOT



When the Logical Not or Logical Invert (NOT) function receives power flow, it sets the state of each bit in the output bit string Q to the opposite of the state of the corresponding bit in bit string IN1.

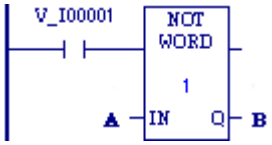
All bits are altered on each scan that power is received, making output string Q the logical complement of input string IN1. Logical NOT passes power flow to the right whenever it receives power. You can specify a string length of 1 to 256 WORDs or DWORDs

Operands

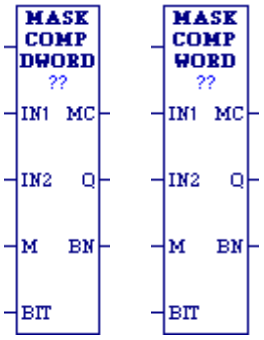
Parameter	Description	Allowed Operands	Optional
Length (??)	The number of WORDs or DWORDs in the bit string to NOT. 1 ≤ Length ≤ 256.	Constant	No
IN1	The first WORD or DWORD of the input string to NOT.	All	No
Q (Must be the same data type as IN1)	The first WORD or DWORD of the NOT's result.	All except constants and variables located in %S memory	No

Example

When input V_I0001 is set, the bit string represented by the variable A is negated. Logical NOT stores the resulting inverse bit string in variable B. Variable A retains its original bit string value.



Masked Compare



The Masked Compare (MASK_COMP_DWORD and MASK_COMP_WORD) function compares the contents of two bit strings. It provides the ability to mask selected bits.

Tip: Input string 1 might contain the states of outputs such as solenoids or motor starters. Input string 2 might contain their input state feedback, such as limit switches or contacts.

When the function receives power flow, it begins comparing the bits in the first string with the corresponding bits in the second string. Comparison continues until a miscompare is found or until the end of the string is reached.

The BIT input stores the bit number where the next comparison should start. Ordinarily, this is the same as the number where the last miscompare occurred. Because the bit number of the last miscompare is stored in output BN, the same reference can be used for both BIT and BN. The comparison actually begins 1 bit following BIT; therefore, the initial value of BIT should be 1 less first bit to be compared (for example, zero (0) to begin comparison at %I00001). Using the same reference for BIT and BN causes the compare to start at the next bit position after a miscompare; or, if all bits compared successfully upon the next invocation of the function, the compare starts at the beginning.

Tip: If you want to start the next comparison at some other location in the string, you can enter different references for BIT and BN. If the value of BIT is a location that is beyond the end of the string, BIT is reset to 0 before starting the next comparison.

The function passes power flow whenever it receives power. The other outputs of the function depend on the state of the corresponding mask bit.

If all corresponding bits in strings IN1 and IN2 match, the function sets the miscompare output MC to 0 and BN to the highest bit number in the input strings. The comparison then stops. On the next invocation of a Masked Compare, it is reset to 0.

If a Miscompare is found, that is, if the two bits being compared are not the same, the function checks the correspondingly numbered bit in string M (the mask).

If the mask bit is a 1, the comparison continues until it reaches another miscompare or the end of the input strings.

If a miscompare is detected and the corresponding mask bit is a 0, the function does the following:

1. Sets the corresponding mask bit in M to 1.
2. Sets the miscompare (MC) output to 1.
3. Updates the output bit string Q to match the new content of mask string M.
4. Sets the bit number output (BN) to the number of the miscompared bit.
5. Stops the comparison.

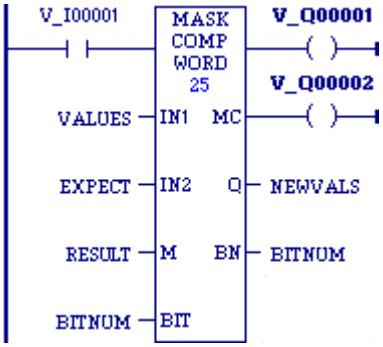
Operands for Masked Compare Function

Parameter	Description	Allowed Operands	Optional
Length (??)	The number of DWORDs or WORDs in the two compared strings. DWORD: 1 ≤ Length ≤ 2,048 WORD: 1 ≤ Length ≤ 4,096	Constant	No
IN1	The first bit string to be compared	All. Constants are legal only when Length is 1	No
IN2	The second bit string to be compared	All. Constants are legal only when Length is 1	No
M	The bit string mask containing the ongoing status of the compare	All except flow or variables in %S memory. Constants are legal only when Length is 1	No
BIT	BIT+1=the bit number where the next comparison starts	All except variables in %S - %SC memories	No
Q	The output copy of the compare mask bit string	All except constants	No
BN	The number of the bit where the latest miscompare occurred, or the highest bit number in the inputs if no miscompare occurred	All except constants and variables in %S memory	No
MC	Can be used to determine if a miscompare has occurred.	flow	Yes

Examples for Masked Compare

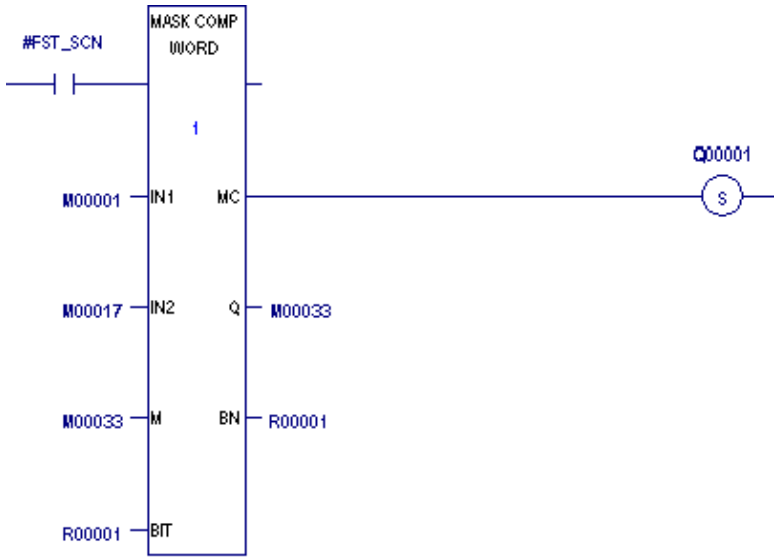
Example 1

When %I00001 is set, MASK_COMP_WORD compares the bits represented by the reference VALUES against the bits represented by the reference EXPECT. Comparison begins at BITNUM+1. If an unmasked miscompare is detected, the comparison stops. The corresponding bit is set in the mask RESULT. BITNUM is updated to contain the bit number of the miscompared bit. In addition, the output string NEWVALS is updated with the new value of RESULT, and coil %Q00002 is turned on. Coil %Q00001 is turned on whenever MASK_COMP_WORD receives power flow.



Example 2

On the first scan, the Masked Compare Word function executes. %M0001 through %M0016 is compared with %M0017 through %M0032. %M0033 through %M0048 contains the mask value. The value in %R0001 determines the bit position in the two input strings where the comparison starts.



Before the function is executed, the contents of the above references are:

(I1) – %M0001 = 6C6Ch =

0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0

(I2) – %M0017 = 606Fh =

0	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

(M/Q) – %M0033 = 000Fh =

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

(BIT/BN) – %R0001 = 0
(MC) – %Q0001 = OFF

The contents of these references after the function block is executed are as follows:

(I1) – %M0001 =

0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0

(I2) – %M0017 =

0	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

(M/Q) – %M0033 =

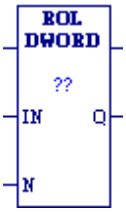
0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1

(BIT/BN) – %R0001 = 8

(MC) – %Q0001 = ON

The #FST_SCN contact forces one and only one execution; otherwise, the function would repeat with possibly unexpected results.

Rotate Bits



Mnemonics:
ROL_DWORD
ROL_WORD
ROR_DWORD
ROR_WORD

When receiving power flow, the Rotate Bits Right (ROR_DWORD and ROR_WORD) and Rotate Bits Left (ROL_DWORD and ROL_WORD) functions rotate all the bits in a string of WORDs or DWORDs N positions respectively to the right or to the left. When rotation occurs, the specified number of bits is rotated out of the input string respectively to the right or to the left and back into the string on the other side.

The Rotate Bits function passes power flow to the right, unless the number of bits to rotate is less than 0, or is greater than the total length of the string. The result is placed in output string Q. If you want the input string to be rotated, the output parameter Q must use the same memory location as the input parameter IN. The entire rotated string is written on each scan that power is received.

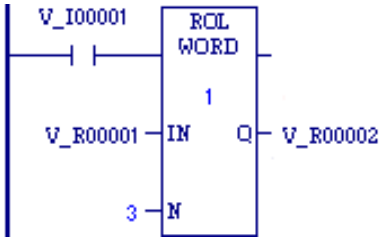
A string length of 1 to 256 words or double words can be specified.

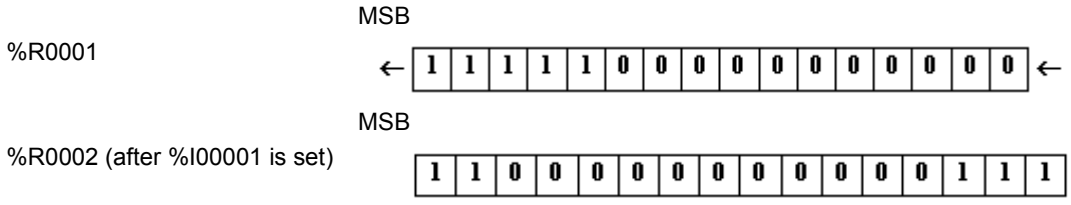
Operands

Parameter	Description	Allowed Operands	Optional
Length (??)	The number of WORDs or DWORDs in the string to be rotated. 1 ≤ Length ≤ 256.	Constant	No
IN	The string to rotate	All. Constants are legal when Length is 1	No
N	The number of positions to rotate. 0 ≤ N ≤ Length.	All except variables in %S - %SC memories	No
Q	The resulting rotated string	All except constants and variables in %S memory	No

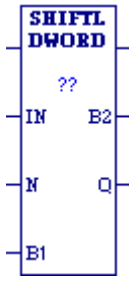
Example

Whenever input V_I0001 is set, the input bit string in location %R0001 is rotated left 3 bits and the result is placed in %R00002. The actual input bit string %R0001 is left unchanged. If the same reference had been used for IN and Q, a rotation would have occurred in place.





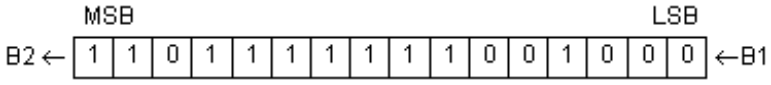
Shift Bits



Mnemonics:
SHIFTL_DWORD
SHIFTL_WORD
SHIFTR_DWORD
SHIFTR_WORD

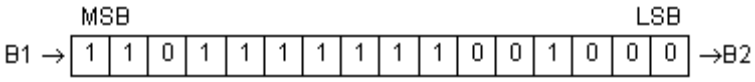
Shift Left

When the Shift Left (SHIFTL_WORD) function receives power flow, it shifts all the bits in a word or group of words to the left by a specified number of places, N. When the shift occurs, the specified number of bits is shifted out of the output string to the left. As bits are shifted out of the high end of the string (Most Significant Bit (MSB)), the same number of bits is shifted in at the low end (Least Significant Bit (LSB)). The SHIFTL_DWORD function operates in a similar manner on DWORDs instead of WORDs.



Shift Right

When the Shift Right (SHIFTR_WORD) function receives power flow, it shifts all the bits in a word or group of words a specified number of places to the right (N). When the shift occurs, the specified number of bits is shifted out of the output string to the right. As bits are shifted out of the low end of the string (LSB), the same number of bits is shifted in at the high end (MSB).



Shift Left and Shift Right

A string length (Length) of 1 to 256 words can be specified.

The bits being shifted into the beginning of the string are specified via input parameter B1. If the value of N is greater than 1, each bit is filled with the same value (0 or 1). This can be:

- The Boolean output of another program function.
- All 1s. To do this, use the #AWL_ON (always on) system bit (in memory location %S7), as a permissive to input B1.
- All 0s. To do this, use the #ALW_OFF (always off) system bit (in memory location %S8), as a permissive to input B1.

The Shift Bits function passes power flow to the right, unless the number of bits specified to shift is zero or is greater than the array size.

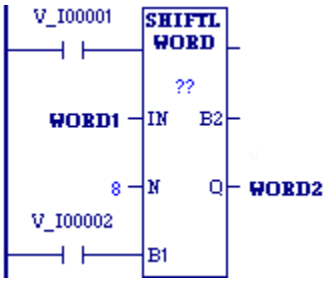
Output Q is the shifted copy of the input string. If you want the input string to be shifted, the output parameter Q must use the same memory location as the input parameter IN. The entire shifted string is written on each scan that power is received. Output B2 is the last bit shifted out. For example, if four bits were shifted, B2 would be the fourth bit shifted out.

Operands

Parameter	Description	Allowed Operands	Optional
Length (??)	The number of WORDs or DWORDs in the string. 1 ≤ Length ≤ 256.	Constants.	No
IN	The string of WORDs or DWORDs to shift	All. Constants are legal only when Length = 1.	No
N	The number of places (bits) to shift the array. 0 ≤ N ≤ Length If N is 0, no shift occurs, but power flow is generated. If N is greater than the number of bits in the string (Length), all bits in Q are set to the value B1, OK is set FALSE, and B2 is set to B1.	All except variables in %S— %SC memories	No
B1	The bit value to shift into the array	flow	No
B2	The bit value of the last bit shifted out of the array.	flow	Yes
Q (Must be the same data type as IN)	The first WORD or DWORD of the shifted array	All except constants and variables in %S memory.	No

Example

Whenever input V_I0001 is set, the bits in the input string that begins at WORD1 are copied to the output bit string that starts at WORD2. WORD2 is left-shifted by 8 bits, as specified by the input N. The resulting open bits at the beginning of the output string are set to the value of V_I0002.



Coils

Coils are used to control the discrete (BOOL) references assigned to them. Conditional logic must be used to control the flow of power to a coil. Coils cause action directly. They do not pass power flow to the right. If additional logic in the program should be executed as a result of the coil condition, you can use an internal reference for the coil or a continuation coil/contact combination.

- A continuation coil does not use an internal reference. It must be followed by a continuation contact at the beginning of any rung following the continuation coil.
- Coils are always located at the rightmost position of a line of logic.

Coil Checking

The level of coil checking is set to **"**Show as error**"** by default. If you want a coil conflict to result in a warning instead of this error, or if you want no warning at all, edit the PLC option: **Multiple Coil Use Warning** in the programming software.

The **"**Show as warning**"** option enables you to use any coil reference with multiple Coils, Set Coils, and Reset Coils, but you will be warned at validation time every time you do so. With both the **"**Show as warning**"** and the **"**no warning**"** options, a reference can be set ON by either a Set Coil or a normal Coil and can be set OFF by a Reset Coil or by a normal Coil.

Graphical Representation of Coils

The programming software displays the COIL, NCCOIL, SETCOIL, and RESETCOIL instructions differently depending on the retentive state of the BOOL variables assigned to them. Examples are provided in the discussion of each type of coil. For a discussion of retentiveness, refer to "Retentiveness of Logic and Data" in chapter 7.

Coil (Normally Open)



A retentive variable is assigned to the coil



A non-retentive variable is assigned to the coil

When a COIL receives power flow, it sets its associated BOOL variable ON (1). When it receives no power flow, it sets the associated BOOL variable OFF (0). COIL can be assigned a retentive variable or a non-retentive variable.

Valid memory areas: %I, %Q, %M, %T, %SA - %SC, and %G. Symbolic discrete variables are permitted. Bit-in-word references on any word-oriented memory except %AI, including symbolic non-discrete memory, are also permitted.

Continuation Coil



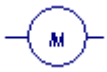
A continuation coil instructs the PLC to continue the present rung's LD logic power flow value (TRUE or FALSE) at the continuation contact on a following rung.

The flow state of the continuation coil is passed to the continuation contact.

Notes:

- If the flow of logic does not execute a continuation coil before it executes a continuation contact, the state of the continuation contact is no flow (FALSE).
- The continuation coil and the continuation contact do not use parameters and do not have associated variables.
- You can have multiple rungs with continuation contacts after a single continuation coil.
- You can have multiple rungs with continuation coils before one rung with a continuation contact.

Negated Coil



A retentive variable is assigned to the negated coil

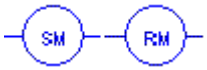


A non-retentive variable is assigned to the negated coil

When it does *not* receive power flow, a negated coil (NCCOIL) sets a discrete reference ON. When it does receive power flow, NCCOIL sets a discrete reference OFF. NCCOIL can be assigned a retentive variable or a non-retentive variable.

Valid memory areas: %I, %Q, %M, %T, %SA - %SC, and %G. Symbolic discrete variables are permitted. Bit-in-word references on any word-oriented memory except %AI, including symbolic non-discrete memory, are also permitted.

Set, Reset Coil



Set Coil and Reset Coil with a retentive variable assigned



Set Coil and Reset Coil with a non-retentive variable assigned

The SET and RESET coils can be used to keep (“latch”) the state of a reference either ON or OFF.

Warning

SET / RESET coils write an undefined result to the transition bit for the given reference. This result differs from that written by Series 90-70 CPUs and could change for future PACSystems CPU models.

Because they write an undefined result to transition bits, do not use SET or RESET coils with references used on POSCON or NEGCON transition contacts.

When a SET coil receives power flow, it sets its discrete reference ON. When a SET coil does not receive power flow, it does not change the value of its discrete reference. Therefore, whether or not the coil itself continues to receive power flow, the reference stays ON until the reference is reset by other logic, such as a RESET coil.

When a RESET coil receives power flow, it resets a discrete reference to OFF. When a RESET coil does not receive power flow, it does not change the value of its discrete reference. Therefore, its reference remains OFF until it is set ON by other logic, such as a SET coil.

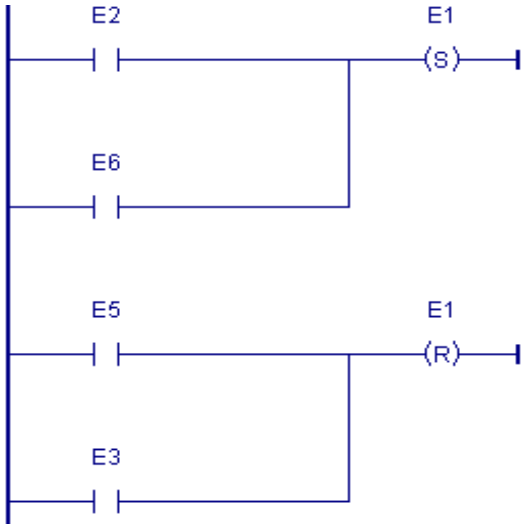
The last solved SET coil or RESET coil of a pair takes precedence.

The SET and RESET coils can be assigned a retentive variable or a non-retentive variable.

Valid memory areas: %I, %Q, %M, %T, %SA - %SC, and %G. Symbolic discrete variables are permitted. Bit-in-word references on any word-oriented memory except %AI, including symbolic non-discrete memory, are also permitted.

Example of Set, Reset Coils

The coil represented by E1 is turned ON when reference E2 or E6 is ON and is turned OFF when reference E5 or E3 is ON.



Transition Coils

The PACSystems provides four transition coils: PTCOIL, NTCOIL, POSCOIL, and NEGCOIL. For examples showing the differences in the operation of the two types of transition coils, see page 8-30.

POSCOIL and NEGCOIL

Positive Transition Coil (POSCOIL) 	***Negative Transition Coil (NEGCOIL)*** 
If: ■ the current value of the *transition* bit for the variable is OFF, ■ the current value of the *status* bit for the variable is OFF, and ■ the current value of the power flow input to the coil is ON, the Positive Transition Coil turns ON the *status* bit of its associated variable. In all other cases, it turns OFF the *status* bit of its associated variable. In all cases, the *transition* bit of the variable is set to the value of the power flow input. **Note:** When the Positive Transition Coil turns ON its reference's *status* bit, it also turns ON its *transition* bit. This negates two of the conditions for the reference bit to be turned ON the next time the Positive Transition coil executes. Therefore the reference bit is turned OFF the next time the Positive Transition Coil executes (as long as the reference bit has not in the meantime been written to by any other logic).	If: ■ the current value of the *transition* bit for the variable is ON, ■ the current value of the *status* bit for the variable is OFF, and ■ the current value of the power flow input is OFF, the Negative Transition Coil turns ON the *status* bit of its associated variable. In all other cases, it turns OFF the *status* bit of its associated variable. In all cases, the *transition* bit of the variable is set to the value of the power flow input. **Note:** When the Negative Transition Coil turns ON its reference's *status* bit, it also turns OFF its *transition* bit. This negates two of the conditions for the reference bit to be turned ON the next time the Negative Transition coil executes. Therefore the reference bit is turned OFF the next time the Negative Transition Coil executes (as long as the reference bit has not in the meantime been written to by any other logic).

Cautions

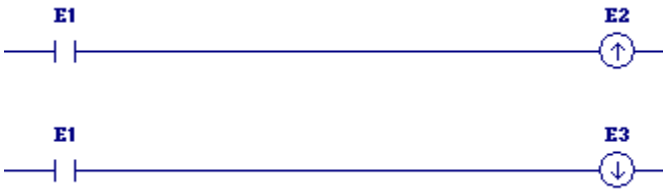
- **Do not override a POSCOIL or NEGCOIL transition coil by putting a force on its reference bit. If a transition coil is overridden and the override is then removed, the behavior of the transition coil on the next sweep in which it is executed depends on many inputs and may be difficult to understand. It may cause unexpected consequences in the ladder logic and in field devices attached to the CPU.**
- **If you want to preserve a transition coil's one-shot nature, do not write to its reference bit using any other instruction, such as another coil or a GE function.**
- **Do not use a transition contact with the same reference address used on a transition coil. The interaction between the two instructions can be difficult to understand.**

Operands for POSCOIL and NEGCOIL

Parameter	Description	Allowed Operands	Optional
BOOL_V	The variable associated with POSCOIL or NEGCOIL	I, Q, M, T, G, SA, SB, SC, and symbolic discrete variables.	No

Example for POSCOIL and NEGCOIL

When reference E1 goes from OFF to ON, coils E2 and E3 receive power flow, turning E2 ON for one logic scan. When E1 goes from ON to OFF, power flow is removed from E2 and E3, turning coil E3 ON for one scan.



PTCOIL and NTCOIL

PTCOILs and NTCOILs behave very similarly to POSCOILs and NEGCOILs. The major difference between them is that PTCOILs and NTCOILs have instance data that is associated with each instance of the coil in logic. The instance data associated with each coil stores the value of the power flow into the coil the last time the coil was executed. Each occurrence of a PTCOIL and NTCOIL in logic has its own copy of instance data. Therefore, two PTCOILs, even if they share the same reference address, operate independently of each other. In contrast, two POSCOILs that share the same reference address affect the behavior of each other.

Because the behavior of a PTCOIL and an NTCOIL is determined solely by the current power flow into the coil and the previous power flow into the coil (i.e., the instance data), it is not affected by writes to its associated BOOL variable by other coils or instructions in the logic. Therefore, many of the cautions that apply to POSCOILs and NEGCOILs do not apply to PTCOILs and NTCOILs.

 **Positive Transition Coil (PTCOIL)**	 **Negative Transition Coil (NTCOIL)**
When the input power flow is ON and the power flow the last time the coil was executed is OFF (i.e., the instance data is OFF), the *status* bit of the BOOL variable associated with PTCOIL is turned ON. Under any other conditions, the *status* bit of the BOOL variable is turned OFF. After the *status* bit of the BOOL variable is updated, the instance data associated with the PTCOIL is set to the value of the input power flow.	When the input power flow is OFF and the power flow the last time the coil was executed is ON (i.e., the instance data is ON), the *status* bit of the BOOL variable associated with NTCOIL is turned ON. Under any other conditions, the *status* bit of the BOOL variable is turned OFF. After the *status* bit of the BOOL variable is updated, the instance data associated with the PTCOIL is set to the value of the input power flow.

Operands for PTCOIL and NTCOIL

Parameter	Description	Allowed Operands	Optional
BOOL_V	The variable associated with PTCOIL or NTCOIL	variables in I, Q, M, T, SA, SB, SC, or G memories as well as symbolic discrete variables. In addition, bit-in-word references on any non-discrete memory (e.g., %R) or on symbolic non-discrete variables are allowed.	No

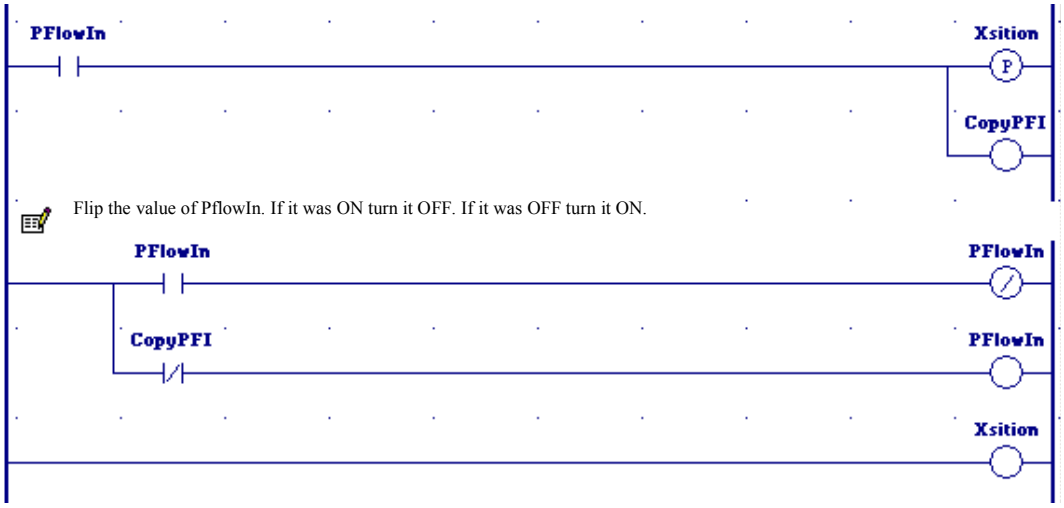
Examples Comparing PTCOIL and POSCOIL

PTCOIL

In the example below, the power flow into the PTCOIL alternates between OFF and ON. On the first sweep the power flow in is OFF, on the second sweep it is ON, and so forth. Each time the power flow into the PTCOIL changes from OFF to ON, the value of Xsition is turned ON. Therefore, on the first sweep, the PTCOIL turns Xsition OFF, on the second sweep it turns it ON, on the third sweep it turns it OFF, and so forth. Notice that the behavior of the PTCOIL is ***not*** affected by the presence of the fourth rung, which also writes to Xsition. PTCOIL behaves the same way when the fourth rung is removed.

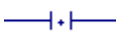
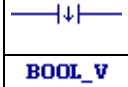
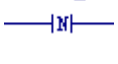
POSCOIL

If a POSCOIL is used in place of the PTCOIL in the example below (keeping the rest of the logic identical and same alternation of power flow into the POSCOIL), the behavior of the logic will be different. The behavior of the POSCOIL ***is*** affected by the execution of the fourth rung, which writes to Xsition and changes both its status and transition bits. In this example, POSCOIL never turns Xsition ON. If the fourth rung is removed, POSCOIL will behave exactly as the PTCOIL behaves, turning Xsition OFF on the first sweep, ON on the second sweep, and so forth.



Contacts

A contact is used to monitor the state of a reference address. Whether the contact passes power flow depends on positive power flow into the contact, the state or status of the reference address being monitored, and the contact type. A reference address is ON if its state is 1; it is OFF if its state is 0.

Contact	*Display*	*Mnemonic*	*Contact Passes Power to Right...*
Continuation Contact		CONTCON	if the preceding continuation coil is set ON
Fault Contact		FAULT	if its associated BOOL or WORD variable has a point fault
High Alarm Contact		HIALR	if the high alarm bit associated with the analog (WORD) reference is ON
Low Alarm Contact		LOALR	if the low alarm bit associated with the analog (WORD) reference is ON
No Fault Contact		NOFLT	if its associated BOOL or WORD variable does not have a point fault
Normally Closed Contact		NCCON	if associated BOOL variable is OFF
Normally Open Contact		NOCON	if associated BOOL variable is ON
Transition Contacts		NEGCON	(negative transition contact) if BOOL reference transitions from ON to OFF
		NTCON	(negative transition contact) if BOOL reference transitions from ON to OFF
		POSCON	(positive transition contact) if BOOL reference transitions from OFF to ON
		PTCON	(positive transition contact) if BOOL reference transitions from OFF to ON

Continuation Contact



- A continuation contact continues the LD logic from the last previously-executed rung in the block that contained a continuation coil.

The flow state of the continuation contact is the same as the preceding executed continuation coil. A continuation contact has no associated variable.

Notes:

- If the flow of logic does not execute a continuation coil before it executes a continuation contact, the state of the continuation contact is no flow.
- The state of the continuation contact is cleared (set to no flow) each time a block begins execution.
- The continuation coil and the continuation contact do not use parameters and do not have associated variables.
- You can have multiple rungs with continuation contacts after a single continuation coil.
- You can have multiple rungs with continuation coils before one rung with a continuation contact.

Fault Contact



A Fault contact (FAULT) detects faults in discrete or analog reference addresses, or locates faults (rack, slot, bus, module).

- To guarantee correct indication of module status, use the reference address (%I, %Q, %AI, %AQ) with the FAULT/NOFLT contacts.
- To locate a fault, use the rack, slot, bus, module fault locating system variable with a FAULT/NOFLT contact.

Note: The fault indication of a given module is cleared when the associated fault is cleared from the fault table.

- For I/O point fault reporting, you must enable point fault references in Hardware Configuration.

FAULT passes power flow if its associated variable or location has a point fault.

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
BWVAR	The variable associated with the FAULT contact	variables in %I, %Q, %AI, and %AQ memories, and predefined fault-locating references	No

High and Low Alarm Contacts



The **high alarm contact** (HIALR) is used to detect a high alarm associated with an analog reference. Use of this contact and the low alarm contact must be enabled during CPU configuration.

A high alarm contact passes power flow if the high alarm bit associated with the analog reference is ON.

The **low alarm contact** (LOALR) detects a low alarm associated with an analog reference. Use of this contact must be enabled during CPU configuration.

A low alarm contact passes power flow if the low alarm bit associated with the analog reference is ON.

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
WORDV	The variable associated with the HIALR or LOALR contact	variables in AI and AQ memories	No

No Fault Contact



A No Fault (NOFLT) contact detects faults in discrete or analog reference addresses, or locates faults (rack, slot, bus, module). NOFLT passes power flow if its associated variable or location does not have a point fault.

- To guarantee correct indication of module status, use the reference address (%I, %Q, %AI, %AQ) with the FAULT/NOFLT contacts.
- To locate a fault, use the rack, slot, bus, module fault locating system variables with a FAULT/NOFLT contact.
- For I/O point fault reporting, you must configure your Hardware Configuration (HWC) to enable the PLC point faults.

Note: The fault indication of a given module is cleared when the associated fault is cleared from the fault table.

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
BWVAR	The variable associated with the NOFLT contact	variables in %I, %Q, %AI, and %AQ memories, and predefined fault-locating references	No

Normally Closed and Normally Open Contacts



A **normally closed contact** (NCCON) acts as a switch that passes power flow if the BOOLV operand is OFF (false, 0).

A **normally open contact** (NOCON) acts as a switch that passes power flow if the BOOLV operand is ON (true, 1).

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
BOOLV	BOOLV may be a predefined system variable or a user-defined variable. NCCON: If BOOLV is ON, the normally closed contact does not pass power flow. If BOOLV is OFF, the contact passes power flow. NOCON: If BOOLV is ON, the normally open contact passes power flow. If BOOLV is OFF, the contact does not pass power flow.	discrete variables in I, Q, M, T, S, SA, SB, SC, and G memories; symbolic discrete variables; bit-in-word references on variables in any non-discrete memory (e.g., %L) or on symbolic non-discrete variables.	No

Transition Contacts

The power flow out of the POSCON and NEGCON transition contacts is determined by the last write to the BOOL variable associated with the contact. The power flow out from the PTCON and NTCON transition contacts is determined by the value that the associated BOOL variable had the last time the contact was executed.

POSCON and NEGCON

BOOLV ⊣↑⊢ ***Positive Transition Contact POSCON***	BOOLV ⊣↓⊢ ***Negative Transition Contact NEGCON***
POSCON passes power flow to the right only when all of the following conditions are met: ■ the input power flow to POSCON is ON ■ the current value of the *status* bit for the associated variable is ON, and ■ the current value of the *transition* bit for the associated variable is ON In other words, if there is positive power flow into a POSCON, and the last time its associated variable was written to, its value went from OFF to ON, the POSCON will pass positive power flow to the right.	NEGCON passes power flow to the right only when all of the following conditions are met: ■ the input power flow to NEGCON is ON ■ the current value of the *status* bit for the associated variable is OFF, and ■ the current value of the *transition* bit for the associated variable is ON In other words, if there is positive power flow into a NEGCON, and the last time its associated variable was written to, its value went from ON to OFF, the NEGCON will pass positive power flow to the right.

Warning

Do not use POSCON or NEGCON transition contacts for references used with transition coils (also called one-shot coils) or SET and RESET coils.

- It is important to note that once a POSCON or NEGCON contact begins passing power flow, it continues to pass power flow until its associated variable is written to. When its variable is written to, regardless of whether the value written to it is ON or OFF, the POSCON or NEGCON contact stops passing power flow.

The source of the write is immaterial; it can be an output coil, a function block output, the input scan, an input interrupt, a data change from the program, or external communications. When the variable is written, the associated POSCON or NEGCON contact is immediately affected. Until a write is made to the variable, the POSCON or NEGCON contact will not be affected.

Depending on the logic flow, writes to the POSCON's or NEGCON's associated variable:

- May occur multiple times during a PLC scan, resulting in the POSCON or NEGCON contact being ON for only a portion of the scan.
- May occur several PLC scans apart, resulting in the POSCON or NEGCON contact being ON for more than one scan.
- May occur once per scan, for example if the POSCON or NEGCON's associated variable is a %I input bit.

An override on a point prevents its status bit from being changed. However, it does not prevent its transition bit from being changed. If a write is attempted to an overridden point, the point's transition bit is cleared. As a result, any associated POSCON or NEGCON contacts will stop passing power flow.

Operands for POSCON and NEGCON

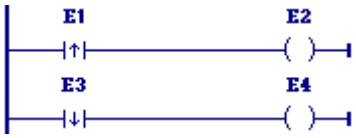
Parameter	Description	Allowed Operands	Optional
BOOLV	The variable associated with the transition contact	variables in I, Q, M, T, S, SA, SB, SC, and G memories, as well as symbolic discrete variables	No

Examples

Example 1

Coil E2 is turned ON when the value of the variable E1 transitions from OFF to ON. It stays ON until E1 is written to again, causing the POSCON to stop passing power flow.

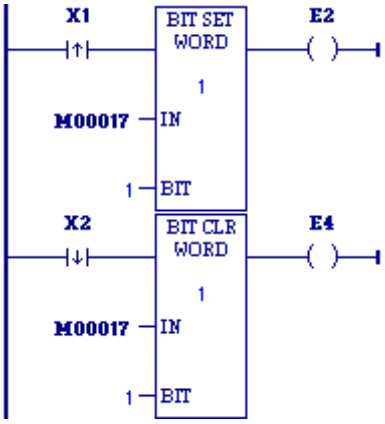
Coil E4 is turned ON when the value of the variable E3 transitions from ON to OFF. It stays ON until E3 is written to again, causing the NEGCON to stop passing power flow.



Example 2

Bit %M00017 is set by a BIT_SET function and then cleared by a BIT_CLR function. The positive transition contact X1 activates the BIT_SET, and the negative transition X2 activates the BIT_CLR.

The positive transition associated with bit %M00017 will be on until %M00017 is reset by the BIT_CLR function. This occurs because the bit is only written when contact X1 goes from OFF to ON. Similarly, the negative transition associated with bit %M00017 will be ON until %M00017 is set by the BIT_SET function.



PTCON and NTCON

The essential difference between the PTCON and NTCON contacts, versus the POSCON and NEGCON contacts, is that each PTCON or NTCON contact instruction used in logic has its own associated instance data. The instance data gives the state (ON or OFF) of the BOOL variable associated with the contact the last time the contact was executed. Because each instance of a PTCON or NTCON instruction has its own instance data, it is possible for two PTCON or NTCON instructions associated with the same BOOL variable to behave differently.

 ***Positive Transition Contact* PTCON**	 ***Negative Transition Contact* NTCON**
PTCON passes power flow to the right only when all of the following conditions are met: ■ The input power flow to PTCON is ON. ■ The current value of the BOOL variable associated with PTCON is ON. ■ The instance data associated with PTCON is OFF (i.e., the value of the associated BOOL variable the last time the PTCON instruction was executed was OFF). After all of these conditions have been evaluated to determine power flow, the PTCON's instance data is updated to contain the current value of the BOOL variable.	NTCON passes power flow to the right only when all of the following conditions are met: ■ The input power flow to NTCON is ON. ■ The current value of the BOOL variable associated with NTCON is OFF. ■ The instance data associated with NTCON is ON (i.e., the value of the associated BOOL variable the last time the NTCON instruction was executed was ON). After all of these conditions have been evaluated to determine power flow, the NTCON's instance data is updated to contain the current value of the BOOL variable.

It is important to note that a PTCON or an NTCON contact will remain ON for exactly one "execution cycle" of the contact. For example, once a PTCON passes power flow, the next time that that particular PTCON instruction is executed, it will NOT pass power flow. This is because the behavior of the PTCON depends on the value of its instance data, which is updated each time the PTCON is executed. When the PTCON is executed and passes power flow, its instance data is updated to contain the current value of its associated BOOL variable, which must be ON. The next time the PTCON is executed, conditions will not be right for it to pass power flow again, because its instance data is ON not OFF.

This behavior is in marked contrast to the behavior of the POSCON and NEGCON contacts, which can pass power flow for many execution cycles at a time.

Also note that because the behavior of the PTCON and NTCON instructions is not dependent on a transition bit, these instructions can be used with variables located in memories that do not have associated transition bits.

Operands for PTCON and POSCON

Parameter	*Description*	*Allowed Operands*	*Optional*
BOOL_V	The variable associated with PTCON or NTCON	variables in I, Q, M, T, S, SA, SB, SC, and G memories, as well as symbolic discrete variables. Also, bit-in-word references on variables in non-discrete memories R, AI, AQ, L, P, W, and on symbolic non-discrete variables.	No

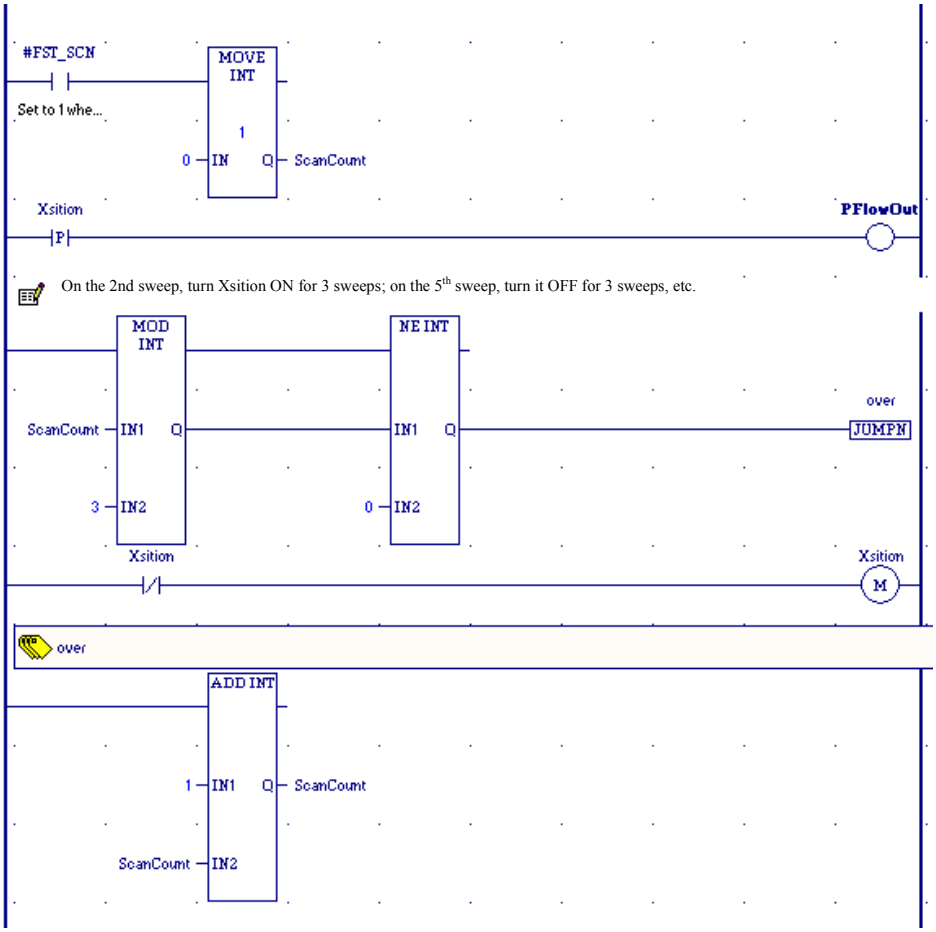
Examples Comparing PTCON and POSCON

PTCON

The logic in the following example starts execution with all variables set to 0. Before the second sweep begins, the Xsition variable used on the PTCON instruction is set to 1. It retains that value for sweeps 2, 3, and 4. Then it is reset back to 0 before sweep 5 begins and retains its 0 value for sweeps 5, 6, and 7. This pattern repeats over and over. The PTCON instruction in rung two passes power flow on the 2nd sweep, the 8th sweep, the 14th sweep, and so on. These are sweeps where the Xsition variable's value becomes a 1, after having been a 0 on the previous sweep. On all other sweeps, the PTCON instruction does not pass power flow.

POSCON

If a POSCON is used in place of the PTCON in the following example (keeping the rest of the logic identical), the same alternation of the Xsition variable's value occurs. The POSCON instruction passes power flow on sweeps 2, 3, and 4; then again on sweeps 8, 9, and 10; and so forth. The POSCON's behavior is dependent on Xsition's transition bit. Since Xsition's value is written once and then simply retained for three sweeps, its transition bit retains its same value for three sweeps. Thus the POSCON will pass or not pass power flow for three sweeps in a row. Note that if Xsition's value is actually written on each sweep, the POSCON and the PTCON behave identically.



Logic Example Using PTCON

Control Functions

The control functions limit program execution and change the way the CPU executes the application program.

Function	*Mnemonic*	*Description*
Do I/O	DO_IO	For one scan, immediately services a specified range of inputs or outputs. (All inputs or outputs on a module are serviced if any reference locations on that module are included in the DO I/O function. Partial I/O module updates are not performed.) Optionally, a copy of the scanned I/O can be placed in internal memory, rather than at the real input points.
Drum	DRUM	Provides predefined On/Off patterns to a set of 16 discrete outputs in the manner of a mechanical drum sequencer.
Edge Detectors	F_TRIG R_TRIG	Detect the changing state of a Boolean signal.
For Loop	FOR_LOOP EXIT_FOR END_FOR	For loop. Repeats the logic between the FOR_LOOP instruction and END_FOR instruction a specified number of times or until EXIT_FOR is encountered.
Mask I/O Interrupt	MASK_IO_INTR	Mask or unmask an interrupt from an I/O module when using I/O variables. If not using I/O variables, use SVC_REQ 17, described in Chapter 10.
Proportional Integral Derivative Control	PID_ISA PID_IND	Provides two PID (Proportional/Integral/Derivative) closed-loop control algorithms: Standard ISA PID algorithm (PID_ISA) Independent term algorithm (PID_IND) **Note:** For details, refer to chapter 11.
Read Switch Position	SWITCH_POS	Reads position of the Run/Stop switch and the mode for which the switch is configured.
Scan Set IO	SCAN_SET_IO	Scans the IO of a specified scan set.
Service Request	SVC_REQ	Requests a special PLC service. **Note:** For details, refer to chapter 10.
Suspend IO	SUS_IO	Suspends for one sweep all normal I/O updates, except those specified by DO I/O instructions.
Suspend or Resume I/O Interrupt	SUSP_IO_INTR	Suspend or resume an I/O interrupt when using I/O variables. If not using I/O variables, use SVC_REQ 32, described in Chapter 10.

Do I/O



When the DO I/O (DO_IO) function receives power flow, it updates inputs or outputs for one scan while the program is running. You can also use DO_ IO to update selected I/O during the program in addition to the normal I/O scan.

You can use DO_IO in conjunction with a Suspend IO (SUS_IO) function, which stops the normal I/O scan. For details, see page 8-55.

If input references are specified, DO_IO allows the most recent values of inputs to be obtained for program logic. If output references are specified, DO I/O updates outputs based on the most current values stored in I/O memory. I/O is serviced in increments of entire I/O modules; the PLC adjusts the references, if necessary, while DO_IO executes. DO_IO does not scan I/O modules that are not configured.

DO_IO continues to execute until all inputs in the selected range have reported or all outputs have been serviced on the I/O modules. Program execution then returns to the function that follows the DO_IO.

If the range of references includes an option module (HSC, APM, etc.), all the input data (%I and %AI) or all the output data (%Q and %AQ) for that module are scanned. The ALT parameter is ignored while scanning option modules.

DO_IO passes power to the right whenever it receives power unless:

- Not all references of the type specified are present within the selected range.
- The CPU is not able to properly handle the temporary list of I/O created by the function.
- The range specified includes I/O modules that are associated with a **"**Loss of I/O**"** fault.

Warning

If DO_IO is used with timed or I/O interrupts, transition contacts associated with scanned inputs may not operate as expected.

Note: The Do I/O function skips modules that do not support DO_IO scanning:

IC693BEM331	90-30 Genius Bus Controller
IC694BEM331	RX3i Genius Bus Controller
IC693BEM341	90-30 2.5 GHz FIP Bus Controller
IC693DNM200	90-30 DeviceNet Master
IC695PBM300	RX3i PROFIBUS Master
IC695PBS301	RX3i PROFIBUS Slave
IC687BEM731	90-70 Genius Bus Controller
IC697BEM731	90-70 Standard Width Genius Bus Controller

Do I/O for Inputs

When DO_IO receives power flow and input references are specified, the PLC scans input points from the starting reference (ST) to the ending reference (END). If a reference is specified for ALT, a copy of the new input values is placed in memory beginning at that reference, and the real input values are not updated. ALT must be the same size as the reference type scanned. If a discrete reference is used for ST and END, ALT must also be discrete.

If no reference is specified for ALT, the real input values are updated. This allows inputs to be scanned one or more times during the program execution portion of the CPU scan.

Do I/O for Outputs

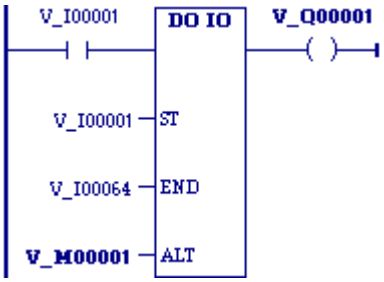
When DO_IO receives power flow and output references are specified, the PLC writes to the output points. If no value is specified in ALT, the range of outputs written to the output modules is specified by the starting reference (ST) and the ending reference (END). If outputs should be written to the output points from internal memory other than %Q or %AQ, the beginning reference is specified for ALT and the end reference is automatically calculated from the length of the END—ST range.

Operands

Parameter	Description	Allowed Operands	Optional
ST	The starting address of the set of input or output points or words to be serviced. ST and END must be in the same memory area. ■ If ST and END are placed in BOOL memory, ST must be byte-aligned. That is, its reference address must start at (8n+1), for example, %I01, %Q09, %Q49. ■ If ST and END are mapped to analog memory, they can have the same reference address. ■ If ST is mapped to an I/O variable, the same I/O variable must also be assigned to the END parameter, and the entire module is scanned.	I, Q, AI, AQ, I/O Variable	No
END	The address of the end bit of input or output points or words to be serviced. Must be in the same memory area as ST. ■ If ST and END are placed in BOOL memory, END's reference address must be 8n, for example, %I08, %Q16. ■ If ST and END are mapped to analog memory, they can have the same reference address. ■ If ST is mapped to an I/O variable, the same I/O variable must also be assigned to the END parameter, and the entire module is scanned.	I, Q, AI, AQ, I/O Variable	No
ALT	For an input scan, ALT specifies the address to store scanned input point/word values. For an output scan, ALT specifies the address to get output point/word values from, to send to the I/O modules. **Note:** ALT can be a WORD only if ST and END are in analog memory.	I, Q, M, T, G, R, AI, AQ	Yes

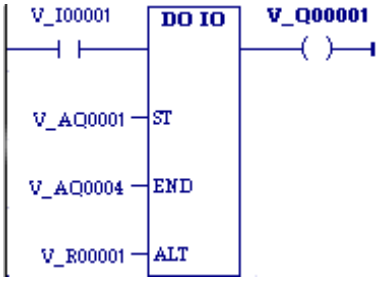
Example - Do I/O for Inputs

When DO_IO receives power flow, the PLC scans references %I0001—64 and %Q0001 is turned on. A copy of the scanned inputs is placed in internal memory from %M0001-64. Because a reference is specified for ALT, the real inputs are not updated. This allows the current values of inputs to be compared with their values at the beginning of the scan. This form of DO_IO allows input points to be scanned one or more times during the program execution portion of the CPU scan.

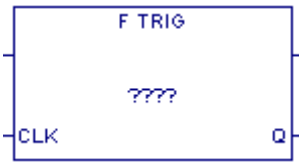


Example - Do I/O For Outputs

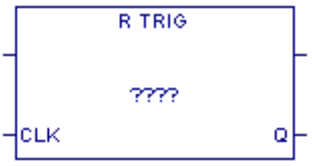
Because a reference is entered for ALT, the values at %AQ001—004 are *not* written to output modules. When DO_IO receives power flow, the PLC writes the values from references %R0001-0004 to the analog output modules and %Q0001 is turned on.



Edge Detectors



Falling Edge Trigger



Rising Edge Trigger

These function blocks detect the changing state of a Boolean signal and produce a single output pulse when an edge is detected.

When transitional instructions, such as Transition Coils (page 8-28) or Transition Contacts (page 8-35), are used inside a function block, there is a problem when the same function block is called more than once per scan. The first call executes the transition correctly but subsequent calls do not because they see the state as adjusted from the first call. The rising and falling edge trigger instructions solve this problem. These instructions have their own instance data that can be a member or an input of the function block so that the transition state follows that of the function block instance and not the function block.

If an edge detector function block is used within a UDFB, its instance data must be a member variable of the UDFB.

Operands

Parameter	Description	Allowed Operands	Optional
????	Instance data for function block. This is a structure variable, described below.	F_TRIG, R_TRIG	No
CLK	Input to be monitored for a change in state.	All	Yes
Q	Edge detection output.	Must be flow in LD. In other languages all types allowed except S, SA, SB, SC and constants.	Yes

Instance Data Structure

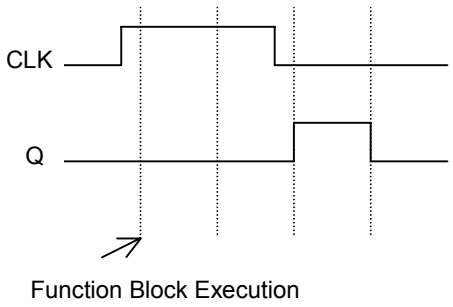
These elements cannot be published or written to.

Element Name	Type	Description
CLK	BOOL	Edge detection input. Not accessible in user logic.
Q	BOOL	Edge detection output. Accessible in user logic. Read only.
STATE	BOOL	Internal value. Not accessible in user logic.
ENO	BOOL	Enable Output. User logic can access as read-only.

F_TRIG Operation

When the CLK input goes from true to false, the output Q is true for one function block execution. The output Q then remains false until a new falling edge is detected.

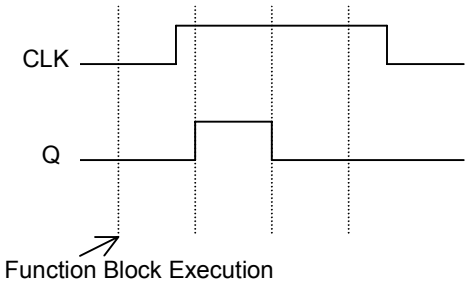
When the PLC transitions from stop to run mode and the CLK input is false and the instance memory is non-retentive, the output Q is true after the function block's first execution. After the next execution, the output is false.



R_TRIG Operation

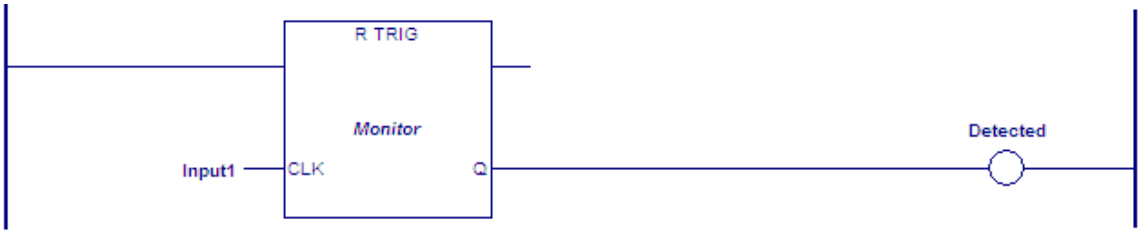
When the CLK input transitions from false to true, the output Q is true for one function block execution. The output Q then remains false until a new rising edge is detected.

When the PLC transitions from stop to run mode and the CLK input is true and the instance memory is non-retentive, the output Q is true after the function block's first execution. After the second execution, the output is false.

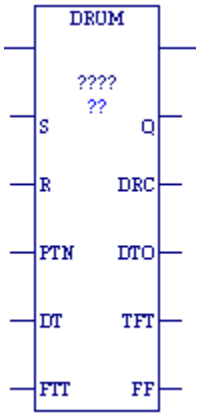


Example

In the following example, when Input1 transitions from OFF to ON, the coil, Detected, is set ON for one function block execution.



Drum



The Drum function operates like a mechanical drum sequencer. The Drum Sequencer steps through a set of potential output bit patterns and selects one based on inputs to the function. The selected value is copied to a group of 16 discrete output references.

When the DRUM function receives power flow, it copies the contents of a selected reference to the Q reference.

Power flow to the R (Reset) input or to the S (Step) input selects the reference to be copied.

The function passes power to the right only if it receives power from the left and no error condition is detected.

The DTO (Dwell Timeout Output) bit is cleared the first time the drum is in a new step. This is true:

- Whether the drum is introduced to a new step by changing the Active Step or by using the S (Step) Input.
- Regardless of the DT (Dwell Time array) value associated with the step (even if it is 0).

During the first sweep the Active Step is initialized.

Operands

Parameter	***Description***	***Allowed Operands***	***Optional***
????	(Control Block) The beginning address of a five-word array that contains the Drum Sequencer's control block. The contents of the control block are described below.	R, P, L, W, Symbolic	No
??	(Length) Value between 1 and 128 that specifies the number of steps.	Constant	No
S	Step input. Used to go one step forward in the sequence. When the function receives power flow and S makes an OFF to ON transition, the Drum Sequencer moves one step. When R (Reset) is active, the function ignores S.	flow	No
R	Reset input. Used to select a specific step in the sequence. When the DRUM function and Reset both receive power flow, DRUM copies the Preset Step value in the Control Block to the Active Step reference in the Control Block. Then the function copies the value in the Preset Step reference to the Q reference bits. When R is active, the function ignores S.	flow	No
PTN	(Pattern) The starting address of an array of words. The number of words is specified by the Length (??) operand. Each word represents one step of the Drum Sequencer. The value of each word represents the desired combination of outputs for a particular value of the Active Step word in the control block. The first element corresponds to an Active Step value of 1; the last element corresponds to an Active Step value of Length. The programming software does not create an array for you. You must ensure you have enough memory for PTN.	All except constant and S, SA—SC numerical data.	No

Parameter	Description	Allowed Operands	Optional
DT	(Dwell Time) If you use the DT operand, you must also use the DTO operand and vice-versa. The DT operand is the starting address of Length words of memory, where Length is the number of steps. Each DT word corresponds to one word of PTN. The value of each word represents the dwell time for the corresponding step of the Drum Sequencer in 0.1 second units. When the dwell time expires for a given step the DTO bit is set. If a Dwell Time is specified, the drum cannot sequence into its next step until the Dwell Time has expired. The programming software does not create an array for you. You must ensure you allocate enough memory for DT.	All except S, SA, SB, SC and constant	Yes
FTT	(Fault Timeout) If you use the FTT operand, you must also use the TFT operand, and vice-versa. The FTT operand is the starting address of Length words of memory, where Length is the number of steps. Each FTT word corresponds to one word of PTN. The value of each word represents the fault timeout for the corresponding step of the Drum Sequencer in 0.1 second units. When the fault timeout has expired the Fault Timeout bit is set. The programming software does not create an array for you. You must ensure you allocate enough memory for FTT.	All except S, SA, SB, SC and constant	Yes
Q	A word of memory containing the element of the PTN that corresponds to the current Active Step.	All except S and constant	No
DRC	(Drum Coil) Bit reference that is set whenever the function is enabled and Active Step is ***not*** equal to Preset Step.	All except S	Yes
DTO	(Dwell Timeout) If you use the DTO operand, you must also use DT and vice-versa. This bit reference is set if the dwell time for the current step has expired.	All except S and constant	Yes
TFT	(Timeout Fault) If you use the TFT operand, you must also use the FTT operand and vice-versa. Bit reference that is set if the drum has been in a particular step longer than the step's specified Fault Timeout.	All except S and constant	Yes
FF	(First Follower) The starting address of (Length/8+1) bytes of memory, where Length is the number of steps. If MOD (Length/8+1)>0, FF has (Length/8+1) bytes. Each bit in the bytes of FF corresponds to one word of PTN. No more than one bit in the FF bytes is ON at any time, and that bit corresponds to the value of the Active Step. The first bit corresponds to an Active Step value of one. The last used bit corresponds to an Active Step value of Length.	All except S and constant	Yes

Control Block for the Drum Sequencer Function

The control block for the Drum Sequencer function contains information needed to operate the Drum Sequencer.

address	Active Step
address + 1	Preset Step
address + 2	Step Control
address + 3	Timer Control

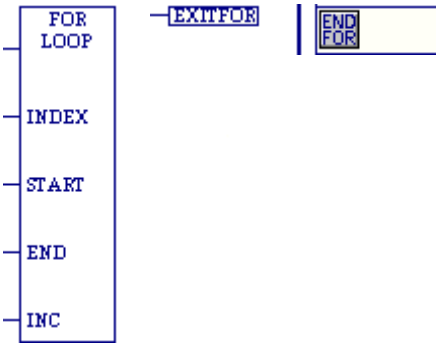
Active Step The active step value specifies the element in the Pattern array to copy to the Out output memory location. This is used as the array index into the Pattern, Dwell Time, Fault Timeout, and First Follower arrays.

Preset Step A word input that is copied to the Active Step output when the Reset is On.

Step Control A word that is used to detect Off to On transitions on both the Step input and the Enable input. The Step Control word is reserved for use by the function, and ***must not be written to.***

Timer Control Two words of data that hold values needed to run the timer. These values are reserved for use by the function and ***must not be written to.***

For Loop



A FOR loop repeats rung logic a specified number of times while varying the value of the INDEX variable in the loop. A FOR loop begins with a FOR_LOOP instruction and ends with an END_FOR instruction. The logic to be repeated must be placed between the FOR and END_FOR instructions. The optional EXIT_FOR instruction enables you to exit the loop if a condition is met before the FOR loop ends normally.

When FOR_LOOP receives power flow, it saves the START, END, and INC (Increment) operands and uses them to evaluate the number of times the rungs between the FOR_LOOP and its END_FOR instructions are executed. Changing the START and END operands while the FOR loop is executing does not affect its operation.

When an END_FOR receives power flow, the FOR loop is terminated and power flow jumps directly to the statement following the END_FOR instruction.

There can be nothing after the FOR_LOOP instruction in the rung and the FOR_LOOP instruction must be the last instruction to be executed in the rung. An EXIT_FOR statement can be placed only between a FOR instruction and an END_FOR instruction. The END_FOR statement must be the only instruction in its rung.

A FOR_LOOP can assign decreasing values to its index variable by setting the increment to a negative number. For example, if the START value is 21, the END value is 1, and the increment value is –5, the statements of the FOR loop are executed five times, and the index variable is decremented by 5 in each pass. The values of the index variable will be 21, 16, 11, 6, and 1.

When the START and END values are set equal, the statements of the FOR loop are executed only once.

When START cannot be incremented or decremented to reach the END, the statements within the FOR loop are not executed. For example, if the value of START is 10, the value of END is 5, and the INCREMENT is 1, power flow jumps directly from the FOR statement to the statement after the END_FOR statement.

Note: If the FOR_LOOP instruction has power flow when it is first tested, the rungs between the FOR and its corresponding END_FOR statement are executed the number of times initially specified by START, END, and INCREMENT. This repeated execution occurs on a single sweep of the PLC and may cause the watchdog timer to expire if the loop is long.

Nesting of FOR loops is allowed, but it is restricted to five FOR/END_FOR pairs. Each FOR instruction must have a matching END_FOR statement following it.

Nesting with JUMPs and MCRs is allowed, provided that they are properly nested. MCRs and ENDMCRs must be completely within or completely outside the scope of a FOR_LOOP/END_FOR pair. JUMPs and LABEL instructions must also be completely within or completely outside the scope of a FOR_LOOP/END_FOR pair. Jumping into or out of the scope of a FOR/END_FOR is not allowed.

Operands

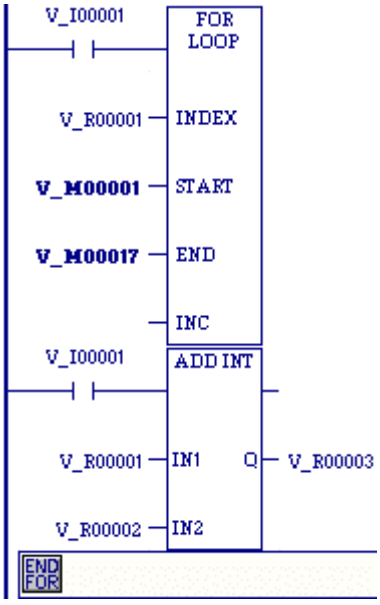
Only the FOR_LOOP function requires operands.

Parameter	***Description***	***Allowed Operands***	***Optional***
INDEX	The index variable. When the loop has completed, this value is undefined. **Note:** Changing the value of the index variable within the scope of the FOR loop is not recommended.	All except constants, flow, and variables in %S - %SC	No
START	The index start value.	All except variables in %S - %SC	No
END	The index end value.	All except variables in %S - %SC	No
INC	The increment value. (Default: 1.)	Constants	Yes

For Loop Examples

Example 1

The value for %M00001 (START) is 1 and the value %M00017 (END) is 10. The INDEX (%R00001) increments by the value of the INC operand (which is assumed to be 1 when omitted) starting at 1 until it reaches the ending value 10. The ADD function of the loop is executed 10 times, adding the current value of (%R00001), which will vary from 1 to 10, to the value I2 (%R00002).



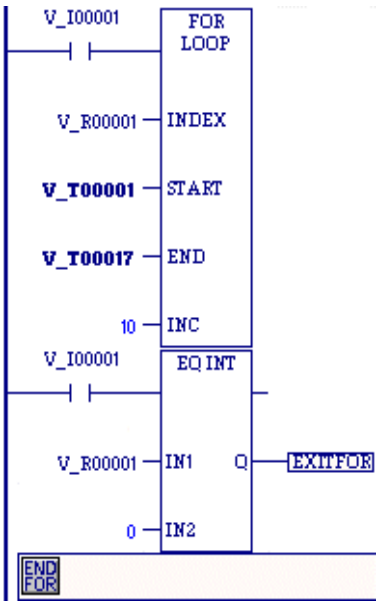
for

I1

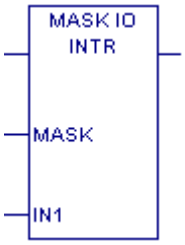
of

Example 2

The value for %T00001 (START) is -100 and the value for %T00017 (END) is 100. The INDEX (%R00001) increments by tens, starting at -100 until it reaches it end value of +100. The EQ function of the loop tries to execute 21 times, with the INDEX (%R00001) being equal to –100, –90, –80, –70, –60, –50, –40, –30, –20, –10, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100. However, when the INDEX (%R00001) is 0, the EXIT statement is enabled and power flow jumps directly to the statement after the END_FOR statement.



Mask I/O Interrupt



Mask or unmask an interrupt from an I/O board when using I/O variables. If not using I/O variables, use SVC_REQ 17.

When the interrupt is masked, the CPU processes the interrupt but does not schedule the associated logic for execution. When the interrupt is unmasked, the CPU processes the interrupt and schedules the associated logic for execution.

When the CPU transitions from Stop to Run, the interrupt is unmasked

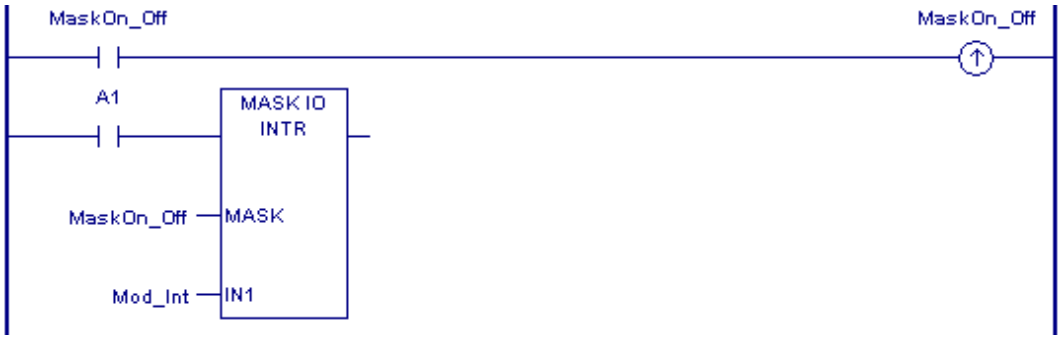
The function passes power to the right when it executes successfully.

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
MASK	Selects unmask or mask operation. Unmask=0; Mask=1	BOOL variable or Bit reference in non-discrete memory	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, AI, AQ, W, symbolic, I/O variable	No
IN1	The interrupt trigger to be masked or unmasked. ■ The I/O board must be a supported input module. ■ The reference address specified must correspond to a valid interrupt trigger reference. ■ The interrupt for the specified channel must be enabled in the configuration.	BOOL or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, I/O variable	No

Example

In the following example, the variable Mod_Int is mapped to an I/O point on a hardware module and is configured as an I/O interrupt to a program block. When the BOOL variable MaskOn_Off transitions from OFF to ON and A1 is set to ON, the interrupt Mod_Int is masked (not executed) for one scan.



Read Switch Position

Read Switch Position (SWITCH_POS) allows the logic to read the current position of the RUN/STOP switch, as well as the mode for which the switch is configured.

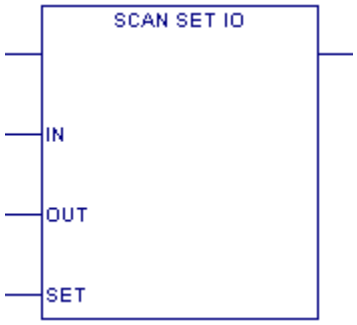


Operands

Parameter	Description	Allowed Operands	Optional
POS	Memory location at which to write current switch position value. 1 - Run I/O Enabled 2 - Run I/O Disabled 3 - Stop Mode	All except S, SA, SB, SC	No
MODE	Memory location to which switch configuration value is written. 0 - Switch configuration not supported 1 - Switch controls run/stop mode 2 - Switch not used, or is used by the user application 3 - Switch controls both memory protection and run/stop mode 4 - Switch controls memory protection	All except S, SA, SB, SC	No

Scan Set IO

The Scan_Set_IO function scans the I/O of a specified scan set number. (Modules can be assigned to scan sets in hardware configuration.) You can specify whether the Inputs and/or Outputs of the associated scan set will be scanned.



Execution of this function block does not affect the normal scanning process of the corresponding scan set. If the corresponding scan set is configured for non-default Number of Sweeps or Output Delay settings, they remain in effect regardless of how many executions of the Scan Set IO function occur in any given sweep.

The Scan Set IO function skips modules that do not support DO_IO scanning (page 8-40.)

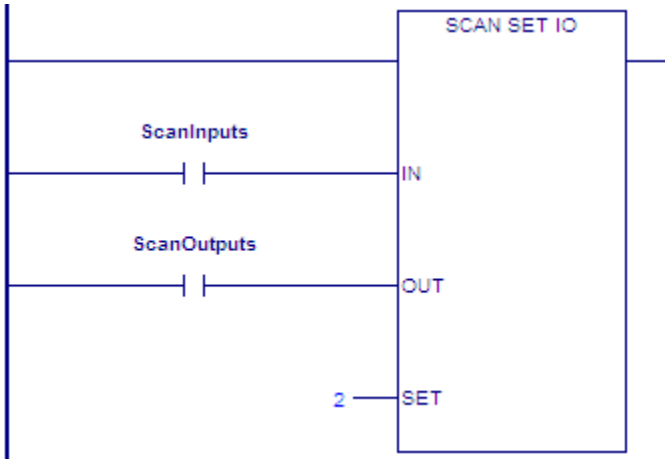
Operands for SCAN_SET_IO

Parameter	Description	Allowed Types	Allowed Operands	Optional
IN	If true the inputs will be scanned.	BOOL variable or bit reference in a non-BOOL variable	Power flow	No
OUT	If true the outputs will be scanned.	BOOL variable or bit reference in a non-BOOL variable	Power flow	No
SET	Number of the scan set to be scanned. Scan sets are specified in the CPU hardware configuration and assigned to modules in the module hardware configuration.	UINT	All except %S memory types.	No
ENO	Energized when all arguments to the function are valid and there are no errors in scanning.	BOOL variable or bit reference in a non-BOOL variable	Power flow.	Yes

Example

By using the Scan Set IO function block in an interrupt block, you can create a custom I/O scan. For example, two Scan Set IO function blocks can be used in an interrupt block to scan the inputs of a scan set at the beginning of the block and the outputs of the same scan set at the end of the block.

In the following example, when ScanInputs is ON, input data for all I/O modules assigned to Scan Set 2 is updated. When ScanOutputs is ON, output data for all I/O modules assigned to Scan Set 2 is updated.



Suspend I/O



The Suspend I/O (SUS_IO) function stops normal I/O scans from occurring for one CPU sweep. During the next output scan, all outputs are held at their current states. During the next input scan, the input references are not updated with data from inputs. However, during the input scan portion of the sweep, the CPU verifies that Genius bus controllers have completed their previous output updates.

Note: SUS_IO function suspends all I/O, both analog and discrete, whether integrated I/O, Genius I/O, or Ethernet Global Data. For details, refer *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.

When SUS_IO receives power flow, all I/O servicing stops except that provided by DO_IO functions.

Warning

If SUS_IO were placed at the left rail of the ladder, without enabling logic to regulate its execution, no regular I/O scan would ever be performed.

SUS_IO passes power flow to the right whenever it receives power.

Example for Suspend I/O

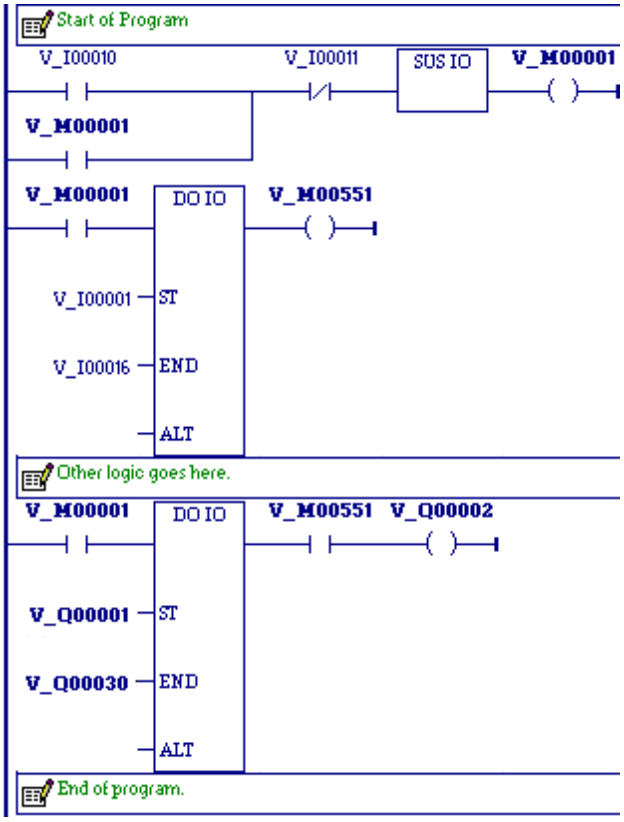
This example shows a SUS_IO function and a DO_IO function used to stop I/O scans, then cause certain I/O to be scanned from the program.

Inputs %I00010 and %I00011 form a latch circuit with the contact from %M00001. This keeps the SUS_IO function active on each sweep until %I00011 goes on. If this input were not scanned by DO_IO after SUS_IO went active, SUS_IO could only be disabled by powering down the PLC.

Output %Q00002 is set when both DO_IO functions execute successfully. The rung is constructed so that both DO_IO functions execute even if one does not set its OK output. With normal I/O suspended, output %Q00002 is not updated until a DO_IO function with %Q00002 in its range executes. This does not occur until the sweep after the setting of %Q00002. Outputs that are set after a DO_IO function executes are not updated until another DO_IO function executes, typically in the next sweep. Because of this delay, most programs that use SUS_IO and DO_IO place the SUS_IO function in the first rung of the program, the DO_IO function that processes inputs in the next rung, and the DO_IO function that processes outputs in the last rung.

The range of the DO_IO function doing outputs is %Q00001 through %Q00030. If the module in this range were a 32-point module, the DO_IO function would actually perform a scan of the entire module. A DO_IO function will not break the scan in the middle of an I/O module.

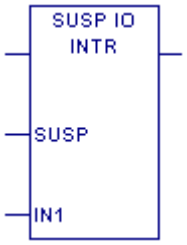
Suspend I/O Sample Logic



Suspend or Resume I/O Interrupt

Suspend or resume an I/O interrupt when using I/O variables.

If not using I/O variables, use SVC_REQ 32.



The function executes successfully and passes power to the right unless:

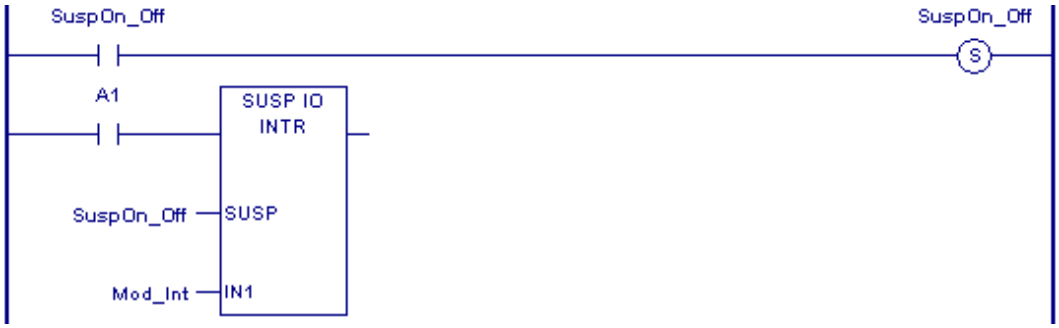
- The I/O module associated with the interrupt trigger specified in IN1 is not supported.
- The reference address specified does not correspond to a valid interrupt trigger reference.
- The specified channel does not have its interrupt enabled in the configuration.

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
SUSP	Selects a suspend or resume operation. 1 (ON)=suspend 0 (OFF)=resume	BOOL variable or bit reference in a non-BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, R, P, L, discrete symbolic, I/O variable	No
IN1	The interrupt trigger to be suspended or resumed.	BOOL or WORD variable	I, Q, M, T, G, R, P, L, AI, AQ, W, I/O variable	No

Example

In the following example, the variable Mod_Int is mapped to an I/O point on a hardware module and is configured as an I/O interrupt to a program block. When the BOOL variable SuspOn_Off is set to ON and A1 is set to ON, interrupts from Mod_Int are suspended until SuspOn_Off is reset.



Conversion Functions

The Conversion functions change a data item from one number format (data type) to another. Many programming instructions, such as math functions, must be used with data of one type. As a result, data conversion is often required before using those instructions.

Function	Description
Convert Angles	
DEG_TO_RAD	Converts degrees to radians
RAD_TO_DEG	Converts radians to degrees
Convert to BCD4 (4-digit Binary-Coded-Decimal)	
UINT_TO_BCD4	Converts UINT (16-bit unsigned integer) to BCD4
INT_TO_BCD4	Converts INT (16-bit signed integer) to BCD4
Convert to BCD8 (8-digit Binary-Coded-Decimal)	
DINT_TO_BCD8	Converts DINT (32-bit signed integer) to BCD8
Convert to INT (16-bit signed integer)	
BCD4_TO_INT	Converts BCD4 to INT
UINT_TO_INT	Converts UINT to INT
DINT_TO_INT	Converts DINT to INT
REAL_TO_INT	Converts REAL to INT
Convert to UINT (16-bit unsigned integer)	
BCD4_TO_UINT	Converts BCD4 to UINT
INT_TO_UINT	Converts INT to UINT
DINT_TO_UINT	Converts DINT to UINT
REAL_TO_UINT	Converts REAL to UINT
Convert to DINT (32-bit signed integer)	
BCD8_TO_DINT	Converts 8-digit Binary-Coded-Decimal (BCD8) to DINT
UINT_TO_DINT	Converts UINT to DINT
INT_TO_DINT	Converts INT to DINT
REAL_TO_DINT	Converts REAL (32-bit signed real or floating-point values) to DINT
LREAL_TO_DINT	Converts REAL (64-bit signed real or floating-point values) to DINT
Convert to REAL (32-bit signed real or floating-point values)	
BCD4_TO_REAL	Converts BCD4 to REAL
BCD8_TO_REAL	Converts BCD8 to REAL
UINT_TO_REAL	Converts UINT to REAL
INT_TO_REAL	Converts INT to REAL
DINT_TO_REAL	Converts DINT to REAL
LREAL_TO_REAL	Converts LREAL to REAL
Convert to LREAL(64-bit signed real or floating-point values)	
DINT_TO_LREAL	Converts DINT to LREAL
REAL_TO_LREAL	Converts REAL to LREAL
Truncate	
TRUNC_DINT	Rounds a REAL number down to a DINT (32-bit signed integer) number
TRUNC_INT	Rounds a REAL number down to an INT (16-bit signed integer) number

Convert Angles



Mnemonics:

DEG_TO_RAD_REAL
DEG_TO_RAD_LREAL
RAD_TO_DEG_REAL
RAD_TO_DEG_LREAL

When the Degrees to Radians (DEG_TO_RAD) or the Radians to Degrees (RAD_TO_DEG) function receives power flow, it performs the appropriate angle conversion on the REAL or LREAL value in input IN and places the result in output Q.

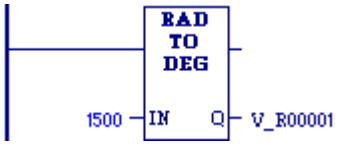
DEG_TO_RAD and RAD_TO_DEG pass power flow to the right when they execute, unless IN is NaN (Not a Number).

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
IN	The value to convert.	All except S, SA, SB, and SC	No
Q	The converted value.	All except S, SA, SB, and SC	No

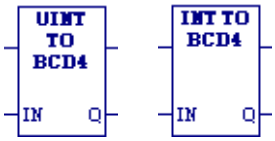
Example

A value of +1500 radians is converted to degrees. The result is placed in %R00001 and %R00002.



Convert UINT or INT to BCD4

When this function receives power flow, it converts the input unsigned (UINT) or signed single-precision integer (INT) data into the equivalent 4-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q.



This function does not change the original input data. The output data can be used directly as input for another program function.

The function passes power flow when power is received, unless the conversion would result in a value that is outside the range 0 to 9,999.

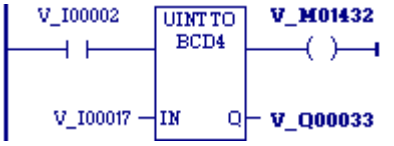
Tip: Data can be converted to BCD format to drive BCD-encoded LED displays or presets to external devices such as high-speed counters.

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
IN	The UINT or INT value to convert to BCD4.	All except S, SA, SB, and SC	No
Q	The BCD4 equivalent value of the original UINT or INT value in IN.	All except S, SA, SB, and SC	No

Example - UINT to BDC4

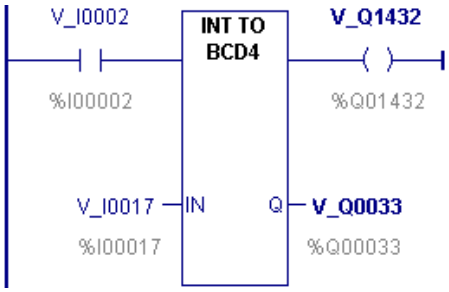
Whenever input %I00002 is set and no errors exist, the UINT at input location %I00017 through %I00032 converted to four BCD digits and the result is stored in memory locations %Q00033 through %Q00048. Coil %M01432 is used to check for successful conversion.



is

Example - INT to BCD4

Whenever input %I0002 is set and no errors exist, the INT values at input locations %I0017 through %I0032 are converted to four BCD digits, and the result is stored in memory locations %Q0033 through %Q0048. Coil %Q1432 is used to check for successful conversion.



Convert DINT to BCD8

When DINT_TO_BCD8 receives power flow, it converts the input signed double-precision integer (DINT) data into the equivalent 8-digit Binary-Coded-Decimal (BCD) values, which it outputs to Q. DINT_TO_BCD8 does not change the original DINT data.



Note: The output data can be used directly as input for another program function.

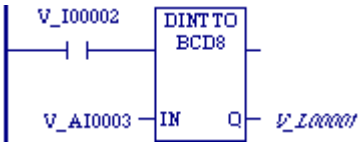
The function passes power flow when power is received, unless the conversion would result in a value that is outside the range 0 to 99,999,999.

Operands

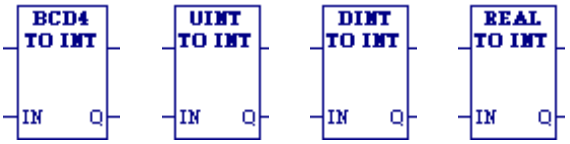
Parameter	*Description*	*Allowed Operands*	*Optional*
IN	The DINT value to convert to BCD8	All except S, SA, SB, and SC	No
Q	The BCD8 equivalent value of the original DINT value in IN	All except S, SA, SB, and SC	No

Example

Whenever input %I00002 is set and no errors exist, the double-precision signed integer (DINT) at input location %AI0003 is converted to eight BCD digits and the result is stored in memory locations %L00001 through %L00002.



Convert BCD4, UINT, DINT, or REAL to INT



BDC4, UINT, and DINT

When this function receives power flow, it converts the input data into the equivalent single-precision signed integer (INT) value, which it outputs to Q. This function does not change the original input data. The output data can be used directly as input for another program function, as in the examples.

The function passes power flow when power is received, unless the data is out of range.

REAL

When REAL_TO_INT receives power flow, it rounds the input REAL data up or down to the nearest single-precision signed integer (INT) value, which it outputs to Q. REAL_TO_INT does not change the original REAL data.

Note: The output data can be used directly as input for another program function.

The function passes power flow when power is received, unless the data is out of range or NaN (Not a Number).

Warning

Converting from REAL to INT may result in overflow. For example, REAL 7.4E15, which equals 7.4 * 10^{15}, converts to INT OVERFLOW.

Tip: To truncate a REAL value and express the result as an INT, i.e., to remove the fractional part of the REAL number and express the remaining integer value as an INT, use TRUNC_INT.

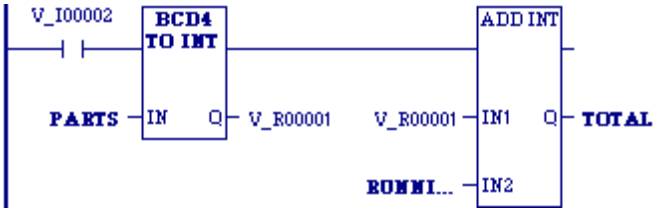
Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
IN	The value to convert to INT.	All except S, SA, SB, and SC	No
Q	The INT equivalent value of the original value in IN.	All except S, SA, SB, and SC	No

Examples

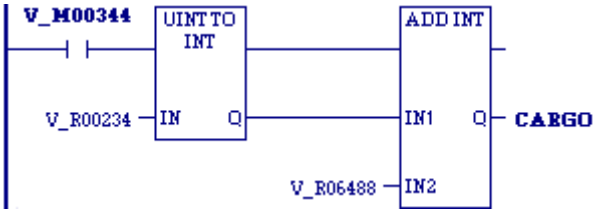
BCD4 to INT

Whenever input %I0002 is set, the BCD-4 value in PARTS is converted to a signed integer (INT) and passed to the ADD_INT function, where it is added to the INT value represented by the reference RUNNING. The sum is output by ADD_INT to the reference TOTAL.



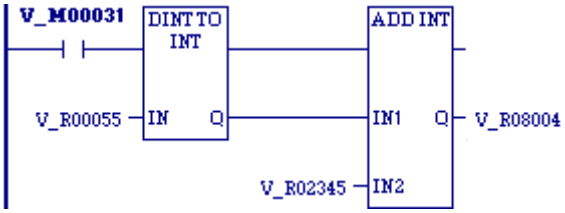
UINT to INT

Whenever input %M00344 is set, the UINT value in %R00234 is converted to a signed integer (INT) and passed to the ADD function, where it is added to the INT value in %R06488. The sum is output by the ADD function to the reference CARGO.

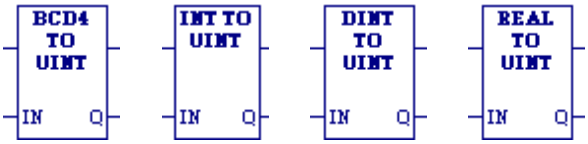


DINT to INT

Whenever input %M00031 is set, the DINT value in %R00055 is converted to a signed integer (INT) and passed to the ADD function, where it is added to the INT at %R02345. The sum is output by the ADD function to %R08004.



Convert BCD4, INT, DINT, or REAL to UINT



When this function receives power flow, it converts the input data into the equivalent single-precision unsigned integer (UINT) value, which it outputs to Q.

The conversion to UINT does not change the original data. The output data can be used directly as input for another program function, as in the example.

The function passes power flow when power is received, unless the resulting data is outside the range 0 to +65,535.

Warning

Converting from REAL to UINT may result in overflow. For example, REAL 7.2E17, which equals 7.2 * 10^{17}, converts to UINT OVERFLOW.

Operands

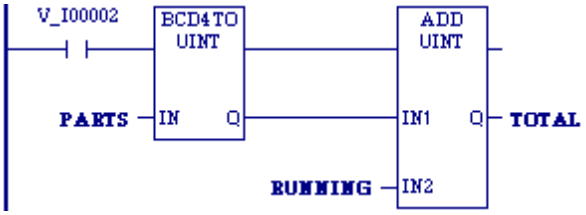
Parameter	Description	Allowed Operands	Optional
IN	The value to convert to UINT.	All except S, SA, SB, and SC	No
Q	The UINT equivalent value of the original input value in IN.	All except S, SA, SB, and SC	No

Examples

BCD4 to UINT

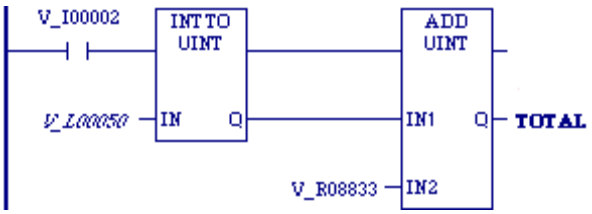
Tip: One use of BCD4_TO_UINT is to convert BCD data from the I/O structure into integer data and store it in memory. This can provide an interface to BCD thumbwheels or external BCD electronics, such as high-speed counters and position encoders.

In the following example, whenever input %I0002 is set, the BCD4 value in PARTS is converted to an unsigned single-precision integer (UINT) and passed to the ADD_UINT function, where it is added to the UINT value represented by the reference RUNNING. The sum is output by ADD_UINT to the reference TOTAL.



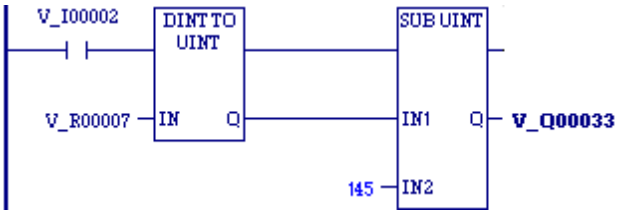
INT to UINT

Whenever input %I0002 is set, the INT value in %L00050 is converted to an unsigned single-precision integer (UINT) and passed to the ADD_UINT function, where it is added to the UINT value in %R08833. The sum is output by ADD_UINT to the reference TOTAL.



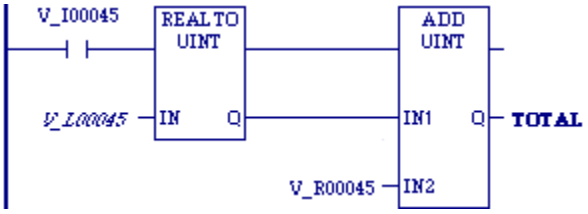
DINT to UINT

Whenever input %I00002 is set and no errors exist, the double precision signed integer (DINT) at input location %R00007 is converted to an unsigned integer (UINT) and passed to the SUB function, where the constant value 145 is subtracted from it. The result of the subtraction is stored in the output reference location %Q00033.

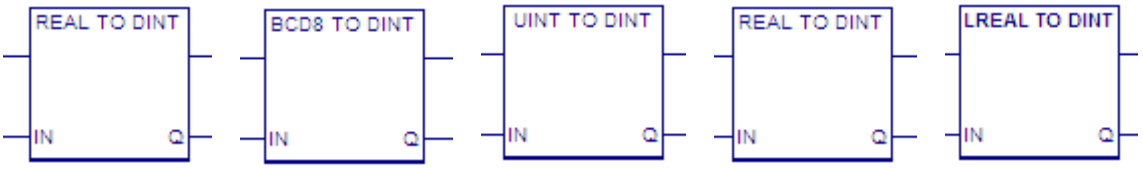


REAL to UINT

Whenever input %I00045 is set, the REAL value in %L00045 is converted to an unsigned single-precision integer (UINT) and passed to the ADD_UINT function, where it is added to the UINT value in %R00045. The sum is output by ADD_UINT to the reference TOTAL.



Convert BCD8, UINT, INT, REAL or LREAL to DINT



BCD8, UINT, and INT

When this function receives power flow, it converts the data into the equivalent signed double-precision integer (DINT) value, which it outputs to Q. The conversion to DINT does not change the original data.

The output data can be used directly as input for another program function. The function passes power flow when power is received, unless the data is out of range.

REAL and LREAL

When REAL_TO_DINT or LREAL_TO_DINT receives power flow, it rounds the input data to the nearest double-precision signed integer (DINT) value, which it outputs to Q. These functions do not change the original REAL or LREAL data.

The output data can be used directly as input for another program function. The function passes power flow when power is received, unless the conversion would result in an out-of-range DINT value.

Warning

Converting from LREAL or REAL to DINT may result in overflow. For example, REAL 5.7E20, which equals 5.7 * 10^{20}, converts to DINT OVERFLOW.

Tip: To truncate a REAL value and express the result as a DINT, i.e., to remove the fractional part of the REAL number and express the remaining integer value as a DINT, use TRUNC_DINT.

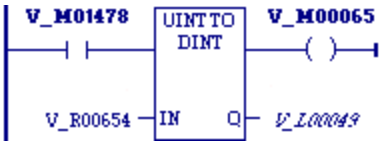
Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
IN	The value to convert to DINT.	All except S, SA, SB, and SC	No
Q	The DINT equivalent value of the original input value in IN.	All except S, SA, SB, and SC	No

Examples

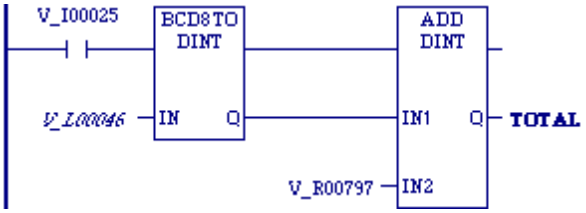
UINT to DINT

Whenever input %M01478 is set, the unsigned single-precision integer (UINT) value at input location %R00654 is converted to a double-precision signed integer (DINT) and the result is placed in location %L00049. The output %M00065 is set whenever the function executes successfully.



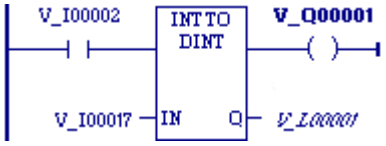
BCD8 to DINT

Whenever input %I00025 is set, the BCD-8 value in %L00046 is converted to a signed double-precision integer (DINT) and passed to the ADD_DINT function, where it is added to the DINT value in %R00797. The sum is output by ADD_DINT to the reference TOTAL.



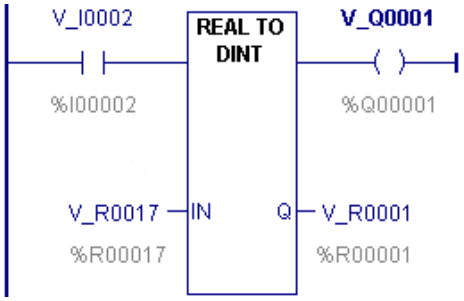
INT to DINT

Whenever input %I00002 is set, the signed single-precision integer (INT) value at input location %I00017 is converted to a double-precision signed integer (DINT) and the result is placed in location %L00001. The output %Q01001 is set whenever the function executes successfully.

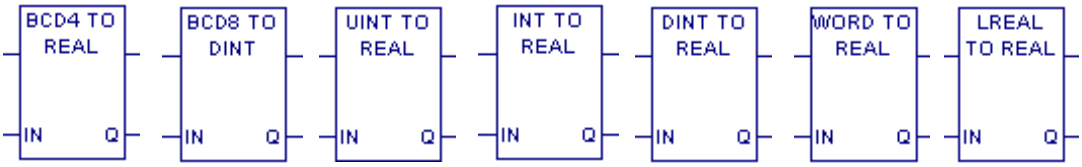


REAL to DINT

Whenever input %I0002 is set, the REAL value at input location %R0017 is converted to a double precision signed integer (DINT) and the result is placed in location %R0001. The output %Q1001 is set whenever the function executes successfully.



Convert BCD4, BCD8, UINT, INT, DINT, and LREAL to REAL



When this function receives power flow, it converts the input data into the equivalent 32-bit floating-point (REAL) value, which it outputs to Q. The conversion to REAL does not change the original input data.

The output data can be used directly as input for another program function.

The function passes power flow when power is received, unless the conversion would result in a value that is out of range.

Warning

Converting from BCD8 to REAL may result in the loss of significant digits.

This is because a BCD8 value is stored in a DWORD, which uses 32 bits to store a value, whereas a REAL (32-bit IEEE floating point number) uses 8 bits to store the exponent and the sign and only 24 bits to store the mantissa.

Warning

Converting from DINT to REAL may result in the loss of significant digits for numbers with more than 7 significant base-10 digits.

This is because a DINT value uses 32 bits to store a value, which is the equivalent of up to 10 significant base-10 digits, whereas a REAL (32-bit IEEE floating point number) uses 8 bits to store the exponent and the sign and only 24 bits to store the mantissa, which is the equivalent of 7 or 8 significant base-10 digits. When the REAL result is displayed as a base-10 number, it may have up to 10 digits, but these are converted from the rounded 24-bit mantissa, so that the last 2 or 3 digits may be inaccurate.

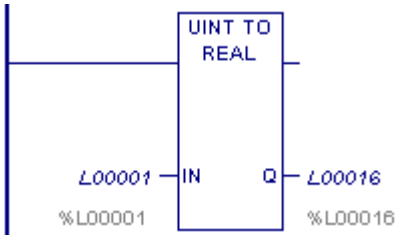
Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
IN	The value to convert to REAL.	All except S, SA, SB, and SC	
Q	The REAL equivalent value of the original input value in IN.	All except S, SA, SB, and SC	

Examples

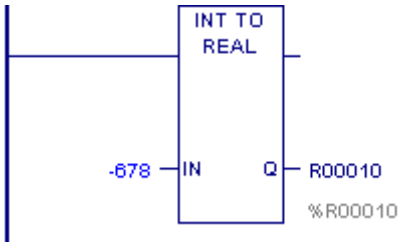
UINT to REAL

The unsigned integer value in %L00001 is 825. The value placed in %L00016 is 825.000.



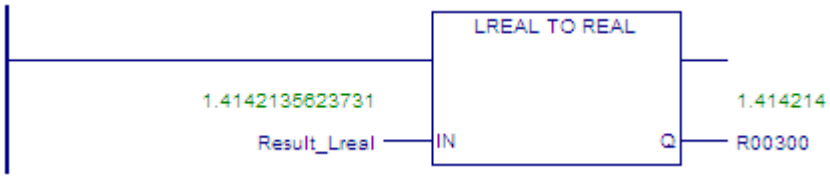
INT to REAL

The integer value of input IN is -678. The value placed in %R00010 is -678.000.



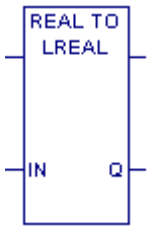
LREAL to REAL

The double-precision floating point value of the square root of 2 is rounded to the nearest single-precision floating point value and placed in R00300.



Convert REAL to LREAL

When REAL_TO_LREAL receives power flow, it converts the 32-bit single precision floating point REAL data to the equivalent 64-bit double-precision floating point data. REAL_TO_LREAL does not change the original REAL data.

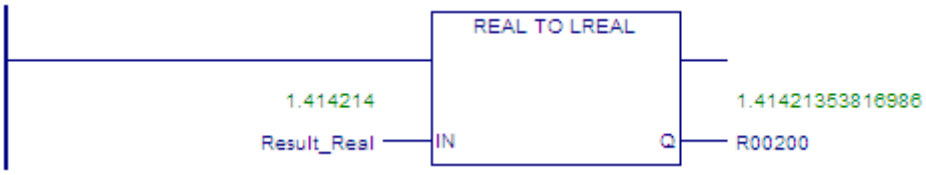


Operands

Parameter	Description	Allowed Operands	Optional
IN	The REAL value to convert to LREAL.	All except S, SA, SB, and SC	No
Q	The LREAL equivalent value of the original REAL value.	All except S, SA, SB, and SC	No

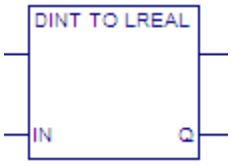
Example

The REAL value of the square root of 2 is converted to the LREAL data type and placed in R00200. Because the actual precision of the data in Result_Real is seven decimal places, the additional decimal places in the data in R00200 are not valid.

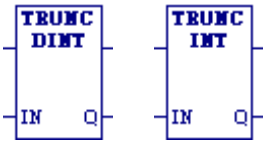


Convert DINT to LREAL

When DINT_TO_LREAL receives power flow, it converts the double-precision input data to 64-bit double-precision floating point data.



Truncate



When power is received, the Truncate functions TRUNC_DINT and TRUNC_INT round a floating-point (REAL) value down respectively to the nearest signed double-precision signed integer (DINT) or signed single-precision integer (INT) value. TRUNC_DINT and TRUNC_INT output the converted value to Q. The original data is not changed.

Note: The output data can be used directly as input for another program function.

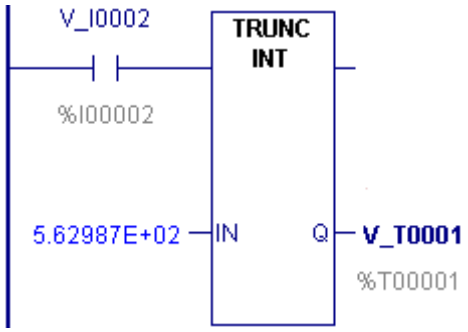
TRUNC_DINT and TRUNC_INT pass power flow when power is received, unless the specified conversion would result in a value that is out of range or unless IN is NaN (Not a Number).

Operands

Parameter	Description	Allowed Operands	Optional
IN	The REAL value whose copy is to be converted and truncated. The original is left intact.	All except S, SA, SB, and SC	No
Q	The truncated value of the original REAL value in IN.	All except S, SA, SB, and SC	No

Example

The displayed constant is truncated and the integer result 562 is placed in %T0001.



Counters

Function	Mnemonic	Description
Down Counter	DNCTR	Counts down from a preset value. The output is ON whenever the Current Value is ≤ 0.
Up Counter	UPCTR	Counts up to a designated value. The output is ON whenever the Current Value is ≥ the Preset Value.

Data Required for Counter Function Blocks

Warning

Do not use two consecutive words (registers) as the starting addresses of two counters. Logic Developer - PLC does not check or warn you if register blocks overlap. Timers will not work if you place the current value of a second timer on top of the preset value for the previous timer.

Each counter uses a one-dimensional, three-word array of %R, %W, %P, %L, or symbolic memory to store the following information:

Current value (CV) Word 1

Warning

The first word (CV) can be read but should not be written to, or the function may not work properly.

Preset value (PV) Word 2 When the Preset Value (PV) operand is a variable, it is normally set to a different location than word 2 in the timer's or counter's three-word array.

- If you use a different address and you change word 2 directly, your change will have no effect, as PV will overwrite word 2.
- If you use the same address for the PV operand and word 2, you can change the Preset Value in word 2 while the timer or counter is running and the change will be effective.

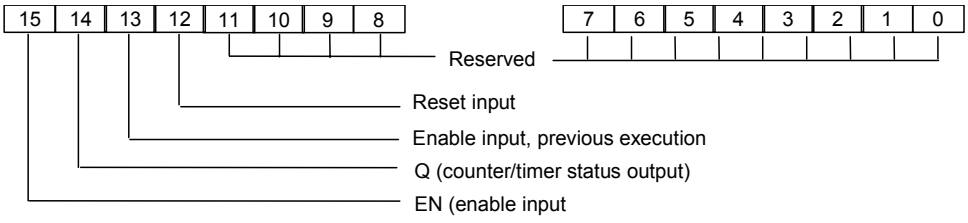
Control word Word 3

Warning

The third word (Control) can be read but should not be written to; otherwise, the function will not work.

The control word stores the state of the Boolean inputs and outputs of its associated timer or counter, as shown in the following diagram:

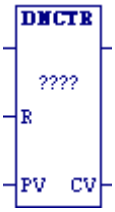
Word 3: Control Word Structure



Note: Bits 0 through 13 are not used for counters.

Down Counter

The Down Counter (DNCTR) function counts down from a preset value. The minimum Preset Value (PV) is zero; the maximum PV is +32,767 counts. When the Current Value (CV) reaches the minimum value, -32,768, it stays there until reset. When DNCTR is reset, CV is set to PV. When the power flow input transitions from OFF to ON, CV is decremented by one. The output is ON whenever CV ≤ 0.



The output state of DNCTR is retentive on power failure; no automatic initialization occurs at power-up.

Warning

Do not use the down counter's Address with other instructions. Overlapping references cause erratic counter operation.

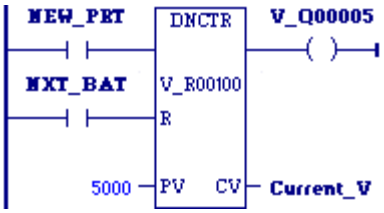
Note: For DNCTR to function properly, you must provide an initial reset to set the CV to the value in PV. If DNCTR is not initially reset, CV will decrement from 0 and the output of DNCTR will be set to ON immediately.

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
Address (????)	The beginning address of a three-word WORD array: Word 1: Current Value (CV) Word 2: Preset Value (PV)% Word 3: Control word	R, W, P, L, symbolic	No
R	When R receives power flow, it resets the counter's CV to PV.	Power flow	No
PV	Preset Value to copy into word 2 of the counter's address when the counter is enabled or reset. 0 ≤PV ≤ 32,767. If PV is out of range, word 2 cannot be reset.	All except S, SA, SB, SC	No
CV	The current value of the counter	All except S, SA, SB, SC and constant	No

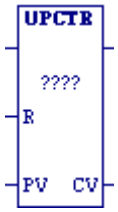
Example – Down Counter

DNCTR counts 5000 new parts before energizing output %Q00005.



Up Counter

The Up Counter (UPCTR) function counts up to the Preset Value (PV). The range is 0 to +32,767 counts. When the Current Value (CV) of the counter reaches 32,767, it remains there until reset. When the UPCTR reset is ON, CV resets to 0. Each time the power flow input transitions from OFF to ON, CV increments by 1. CV can be incremented past the Preset Value (PV). The output is ON whenever CV ≥ PV. The output (Q) stays ON until the R input receives power flow to reset CV to zero.



The state of UPCTR is retentive on power failure; no automatic initialization occurs at powerup.

Operands

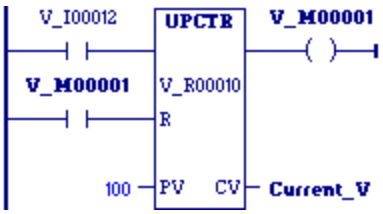
Warning

Do not use the up counter's Address with other instructions. Overlapping references cause erratic counter operation.

Parameter	*Description*	*Allowed Operands*	*Optional*
Address (????)	The beginning address of a three-word WORD array: Word 1: Current Value (CV) Word 2: Preset Value (PV) Word 3: Control word	R, W, P, L, symbolic	No
R	When R is ON, it resets the counter's CV to 0.	Power flow	No
PV	Preset Value to copy into word 2 of the counter's address when the counter is enabled or reset. 0 ≤ PV ≤ 32,767. If PV is out of range, it does not affect word 2.	All except S, SA, SB, and SC	No
CV	The current value of the counter	All except S, SA, SB, SC and constant	No

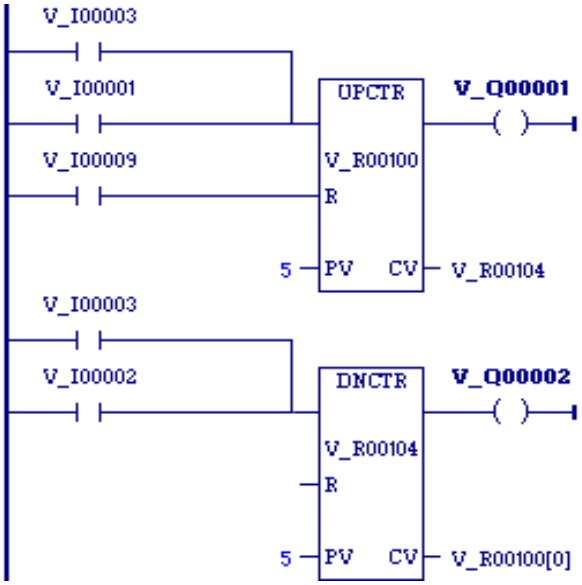
Example – Up Counter

Every time input %I0012 transitions from OFF to ON, the Up Counter counts up by 1; internal coil %M0001 is energized whenever 100 parts have been counted. Whenever %M0001 is ON, the accumulated count is reset to zero.



Example – Up Counter and Down Counter

This example uses an up/down counter pair with a shared register for the accumulated or current value. When the parts enter the storage area, the up counter increments by 1, increasing the current value of the parts in storage by a value of 1. When a part leaves the storage area, the down counter decrements by 1, decreasing the inventory storage value by 1. To avoid conflict with the shared register, both counters use different register addresses but each has a current value (CV) address that is the same as the accumulated value for the other register.



Data Move Functions

The Data Move functions provide basic data move capabilities.

Function	*Mnemonics*	*Description*
Block Clear	BLK_CLR_WORD	Replaces all the contents of a block of data with zeros. Can be used to clear an area of WORD or analog memory.
Block Move	BLKMOV_DINT BLKMOV_DWORD BLKMOV_INT BLKMOV_REAL BLKMOV_UINT BLKMOV_WORD	Copies a block of seven constants to a specified memory location. The constants are input as part of the function.
Communication Request	COMM_REQ	Allows the program to communicate with an intelligent module, such as a Genius Bus Controller or a High Speed Counter.
Data Initialization	DATA_INIT_DINT DATA_INIT_DWORD DATA_INIT_INT DATA_INIT_REAL DATA_INIT_LREAL DATA_INIT_UINT DATA_INIT_WORD	Copies a block of constant data to a reference range. The mnemonic specifies the data type.
Data Initialize ASCII	DATA_INIT_ASCII	Copies a block of constant ASCII text to a reference range.
Data Initialize DLAN	DATA_INIT_DLAN	Used with a DLAN Interface module.
Data Initialize Communications Request	DATA_INIT_COMM	Initializes a COMM_REQ function with a block of constant data. The length should equal the size of the COMM_REQ function's entire command block.
Move Data	MOVE_BOOL MOVE_DINT MOVE_DWORD MOVE_INT MOVE_REAL MOVE_LREAL MOVE_UINT MOVE_WORD	Copies data as individual bits, so the new location does not have to be the same data type. Data can be moved into a different data type without prior conversion.
Shift Register	SHFR_BIT SHFR_DWORD SHFR_WORD	Shifts one or more data bits, data WORDs or data DWORDs from a reference location into a specified area of memory. Data already in the area is shifted out.
Swap	SWAP_DWORD SWAP_WORD	Swaps two BYTEs of data within a WORD or two WORDs within a DWORD.
Bus Read	BUS_RD_BYTE BUS_RD_DWORD BUS_RD_WORD	Reads data from a module on the bus.
Bus Read Modify Write	BUS_RMW_BYTE BUS_RMW_DWORD BUS_RMW_WORD	Uses a read/modify/write cycle to update a data element in a module on the bus.
Bus Test and Set	BUS_TS_BYTE BUS_TS_WORD	Handles semaphores on the bus.
Bus Write	BUS_WRT_BYTE BUS_WRT_DWORD BUS_WRT_WORD	Writes data to a module on the bus.

Block Clear

When the Block Clear (BLKCLR_WORD) function receives power flow, it fills the specified block of data with zeros, beginning at the reference specified by IN. When the data to be cleared is from BOOL (discrete) memory (%I, %Q, %M, %G, or %T), the transition information associated with the references is updated. BLKCLR_WORD passes power to the right whenever it receives power.



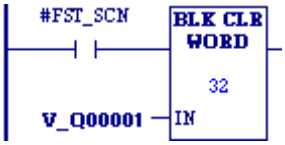
Note: The input parameter IN is not included in coil checking.

Operands

Parameter	Description	Allowed Operands	Optional
Length (??)	The number of words to clear, starting at the IN location. 1 ≤ Length ≤ 256 words.	Constant	No
IN	The first WORD of the memory block to clear to 0.	All except %S and data flow.	No

Example

At power-up, 32 words of %Q memory (512 points) beginning at %Q0001 are filled with zeros. The transition information associated with these references will also be updated.



Block Move

When the Block Move (BLKMOV) function receives power flow, it copies a block of seven constants into consecutive locations beginning at the destination specified in output Q. BLKMOV passes power to the right whenever it receives power.



Mnemonics:
BLKMOV_DINT
BLKMOV_DWORD
BLKMOV_INT
BLKMOV_REAL
BLKMOV_UINT
BLKMOV_WORD

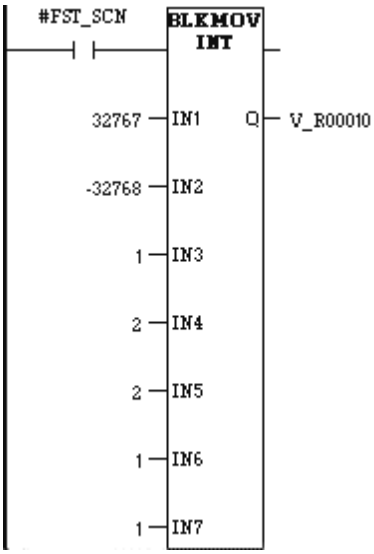
Operands

Note: For each mnemonic, use the corresponding data type for the Q operand. For example, BLKMOV_DINT requires Q to be a DINT variable.

Parameter	*Description*	*Allowed Operands*	*Optional*
IN1 to IN7	The seven constant values to move.	Constants. Constant type must match function type.	No
Q	The first memory location of the destination for the moved values. IN1 is moved to Q.	All except %S. %SA, SB, SC are also prohibited on BLKMOV REAL, BLK_MOV_INT, and BLK_MOV_UINT.	No

Example

When the enabling input represented by the name #FST_SCN is ON, BLKMOV_INT copies the seven input constants into memory locations %R0010 through %R0016.



BUS_ Functions

Four program functions allow the PACSystems CPU to communicate with modules in the system.

- Bus Read (BUS_RD)
- Bus Write (BUS_WRT)
- Bus Read/Modify/Write (BUS_RMW)
- Bus Test and Set (BUS_TS)

These functions use the same parameters to specify which module on the bus will exchange data with the CPU.

Note: Additional information related to addressing modules is required to use the BUS_ functions. For non-GE Fanuc VME modules in an RX7i system, refer to the *PACSystems RX7i User's Guide to Integration of VME Modules,* GFK-2235. For other non-GE Fanuc modules, refer to the product documentation provided by the manufacturer.

Rack, Slot, Subslot, Region, and Offset Parameters

The rack and slot parameters refer to a module in the hardware configuration. The region parameter refers to a memory region configured for that module. The subslot is ordinarily set to 0. The offset is a 0-based number that the function adds to the module's base address (which is part of the memory region configuration) to compute the address to be read or written.

BUS Read

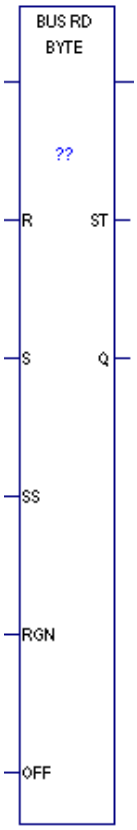
The BUS_RD function reads data from the bus. This function should be executed before the data is needed in the program. If the amount of data to be read is greater than 32767 BYTES, WORDS, or DWORDS, use multiple instructions to read the data.

When BUS_RD receives power flow, it accesses the module at the specified rack (R), slot (S), subslot (SS), address region (RGN) and offset (OFF). BUS_RD copies the specified number (Length) of data units (DWORDS, WORDs or BYTEs) from the module to the CPU, beginning at output reference (Q).

The function passes power to the right when its operation is successful. The status of the operation is reported in the status location (ST).

Note: For each BUS_RD function type, use the corresponding data type for the Q operand. For example, BUS_RD_BYTE requires Q to be a BYTE variable.

Note: An interrupt block can preempt the execution of a BUS_RD function. On the bus, only 256 bytes are read coherently (i.e., read without being preempted by an interrupt).



Mnemonics:
BUS_RD_DINT
BUS_RD_DWORD
BUS_RD_WORD

Operands for BUS READ

Parameter	Description	Allowed Operands	Optional
Length (??)	The number of BYTEs, DWORDs, or WORDs. 1 to 32,767.	Constant	No
R	Rack number. UINT constant or variable.	All except %S—%SC	No
S	Slot number. UINT constant or variable.	All except %S—%SC	No
SS	Subslot number (defaults to 0). UINT constant or variable.	All except %S—%SC	Yes
RGN	Region (defaults to 1). WORD constant or variable.	All except %S—%SC	Yes
OFF	The offset in bytes. DWORD constant or variable.	All except %S—%SC	No
ST	The status of the operation. WORD variable.	All except variables located in %S—%SC, and constants	Yes
Q	Reference for data read from the module. DWORD variable.	All except variables located in %S—%SC, and constants	No

BUS_RD Status in the ST Output

The BUS_RD function returns one of the following values to the ST output:

0	Operation successful.
1	Bus error
2	Module does not exist at rack/slot location.
3	Module at rack/slot location is an invalid type.
4	Start address outside the configured range.
5	End address outside the configured address range.
6	Absolute address even but interface configured as odd byte only
8	Region not enabled
10	Function parameter invalid.

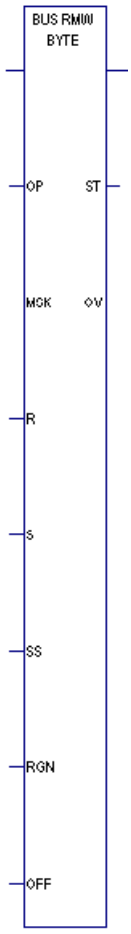
BUS Read Modify Write

The BUS_RMW function updates one byte, word, or double word of data on the bus. This function locks the bus while performing the read-modify-write operation.

When the BUS_RMW function receives power flow through its enable input, the function reads a dword, word or byte of data from the module at the specified rack (R), slot (S), subslot (SS) and optional address region (RGN) and offset (OFF). The original value is stored in parameter (OV).

The function combines the data with the data mask (MSK). The operation performed (AND / OR) is selected with the OP parameter. The mask value is dword data. When operating on a word of data, only the lower 16 bits are used. When operating on a byte of data, only the lower 8 bits of the mask data are used. The result is then written back to the same address from which it was read.

The BUS_RMW function passes power to the right when its operation is successful, and returns a status value to the ST output.



Other mnemonic:
BUS_RMW_WORD

Operands for BUS_RMW

For BUS_RMW_WORD, the absolute bus address must be a multiple of 2. For BUS_RMW_DWORD, it must be a multiple of 4.

The absolute bus address is equal to the base address plus the offset value.

Parameter	***Description***	***Allowed Operands***	***Optional***
OP	Type of operation: 0 = AND 1 = OR	Constant	No
MSK	The data mask. DWORD constant or variable.	All except %S—%SC	No
R	Rack number. UINT constant or variable.	All except %S—%SC	No
S	Slot number. UINT constant or variable.	All except %S—%SC	No
SS	Subslot number (optional, defaults to 0). UINT constant or variable.	All except %S—%SC	Yes
RGN	Region (defaults to 1). WORD constant or variable.	All except %S—%SC	Yes
OFF	The offset in bytes. DWORD constant or variable.	All except %S—%SC	No
ST	The status of the operation. WORD variable.	All except variables located in %S—%SC, and constants	Yes
OV	Original value. DWORD variable.	All except variables located in %S—%SC, and constants	Yes

BUS_RMW Status in the ST Output

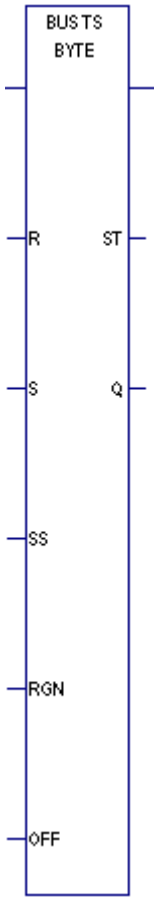
The BUS_RMW function returns one of the following values to the ST output:

0	Operation successful.
1	Bus error
2	Module does not exist at rack/slot location.
3	Module at rack/slot location is an invalid type.
4	Start address outside the configured range.
5	End address outside the configured address range.
6	Absolute address even but interface configured as odd byte only
7	For WORD type, absolute bus address is not a multiple of 2. For DWORD type, absolute bus address is not a multiple of 4.
8	Region not enabled
9	Function type too large for configured access type.
10	Function parameter invalid.

BUS Test and Set

The BUS_TS function uses semaphores to control access to specific memory in a module located on the bus. The BUS_TS function exchanges a Boolean TRUE (1) for the value currently at the semaphore location. If that value was already a 1, then the BUSTST function does not acquire the semaphore. If the existing value was 0, the semaphore is set and the BUS_TS function has the semaphore and the use of the memory area it controls. The semaphore can be cleared and ownership relinquished by using the BUSWRT function to write a 0 to the semaphore location. This function locks the bus while performing the operation.

When the BUS_TS function receives power flow through its enable input, the function exchanges a Boolean TRUE (1) with the address specified by the RACK, SLOT, SUBSLOT, RGN, and OFF parameters. The function sets the Q output on if the semaphore was available (0) and was acquired. It passes power flow to the right whenever power is received and no errors occur during execution.



Other mnemonic:
BUS_TS_WORD

Operands for BUS Test and Set

BUS_TS can be programmed as BUS_TS_BYTE or BUS_TS_WORD. For BUS_TS_WORD, the absolute address of the module must be a multiple of 2. The absolute address is equal to the base address plus the offset value.

Parameter	*Description*	*Allowed Operands*	*Optional*
R	Rack number. UINT constant or variable.	All except %S—%SC	No
S	Slot number. UINT constant or variable.	All except %S—%SC	No
SS	Subslot number (defaults to 0). UINT constant or variable.	All except %S—%SC	Yes
RGN	Region (defaults to 1). WORD constant or variable.	All except %S—%SC	Yes
OFF	The offset in bytes. DWORD constant or variable.	All except %S—%SC	No
ST	The status of the bus test and set operation. WORD variable.	All except variables located in %S—%SC, and constant	Yes
Q	Output set on if the semaphore was available (0). Otherwise, Q is set off.	Power flow	Yes

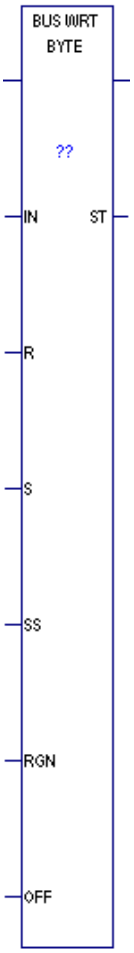
BUS Write

When the BUS_WRT function receives power flow through its enable input, it writes the data located at reference (IN) to the module at the specified rack (R), slot (S), subslot (SS) and optional address region (RGN) and offset (OFF). BUSWRT writes the specified length (LEN) of data units (DWORDS, WORDs or BYTEs).

The BUS_WRT function passes power to the right when its operation is successful. The status of the operation is reported in the status location (ST).

Note: For each BUS_WRT function type, use the corresponding data type for the IN operand. For example, BUS_WRT_BYTE requires IN to be a BYTE variable.

Note: An interrupt block can preempt the execution of a BUS_WRT function. On the bus, only 256 bytes are written coherently (i.e., written without being preempted by an interrupt).

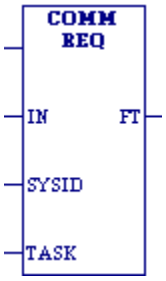


Mnemonics:
BUS_WRT_DINT
BUS_WRT_DWORD
BUS_WRT_WORD

Operands for Bus Write

Parameter	*Description*	*Allowed*	*Optional*
Length (??)	Length. The number of BYTEs, DWORDs, or WORDs. 1 to 32,767.	Constant	No
IN	Reference for data to be written to the module. DWORD variable.	All except variables located in %S—%SC, and constant	No
R	Rack number. UINT constant or variable.	All except %S—%SC	No
S	Slot number. UINT constant or variable.	All except %S—%SC	No
SS	Subslot number (defaults to 0) UINT constant or variable.	All except %S—%SC	Yes
RGN	Region. (defaults to 1) WORD constant or variable.	All except %S—%SC	Yes
OFF	The offset in bytes. DWORD constant or variable.	All except %S—%SC	No
ST	The status of the operation. WORD variable.	All except variables located in %S—%SC, and constant	Yes

Communication Request



The Communication Request (COMM_REQ) function communicates with a GE Fanuc intelligent module, such as a Genius Communications Module or High Speed Counter.

Notes:

- The information presented in this section shows only the basic format of the COMM_REQ function. Many types of COMM_REQs have been defined. You will need additional information to program the COMM_REQ for each type of device. Programming requirements for each module that uses the COMM_REQ function are described in the specialty module's user documentation.
- If you are using serial communications, refer to chapter 14, “Serial I/O, SNP and RTU Protocols.”
- A COMM_REQ instruction inside an interrupt block being executed may cause the block to be preempted when a new, incoming interrupt has the same priority.

When COMM_REQ receives power flow, it sends the command block of data specified by the IN operand to the communications TASK in the intelligent or specialty module, at the rack/slot location specified by the SYSID operand. The command block contents are sent to the receiving device and the program execution resumes immediately. (Because PACSystems does not support WAIT mode COMM_REQs, the timeout value is ignored.)

The COMM_REQ passes power flow unless the following fault conditions exist. The Function Faulted (FT) output may be set ON if:

- Control block is invalid
- Destination is invalid (target module is not present or is faulted)
- Target module cannot receive mail because its queue is full

The Function Faulted output may have these states:

Enable	*Error?*	*Function Faulted Output*
active	no	OFF
active	yes	ON
not active	no execution	OFF

Command Block

The command block provides information to the intelligent module on the command to be performed. The command block starts at the reference specified by the operand IN. This address may be in any word-oriented area of memory (%R, %P, %L, %W, %AI, %AQ, or symbolic non-discrete variables). The length of the command block depends on the amount of data sent to the device.

The Command Block contains the data to be communicated to the other device, plus information related to the execution of the COMM_REQ. Information required for the command block can be placed in the designated memory area using a programming function such as MOVE, BLKMOV, or DATA_INIT_COMM.

Command Block Structure

Address	Data Block Length (in words)	The number of data words starting with the data at address+6 to the end of the command block, inclusive. The data block length ranges from 1 to 128 words. Each COMM_REQ command has its own data block length. When entering the data block length, you must ensure that the command block fits within the register limits
Address + 1	Wait/No Wait Flag	Must be set to 0 (No Wait)
Address + 2	Status Pointer Memory Type	Specifies the memory type for the location where the COMM_REQ status word (CSR) returned by the device will be written when the COMM_REQ completes.
Address + 3	Status Pointer Offset	The word at address + 3 contains the offset for the status word within the selected memory type. **Note:** The status pointer offset is a zero-based value. For example, %R00001is at offset zero in the register table.
Address + 4	Idle Timeout Value	This parameter is ignored in No Wait mode.
Address + 5	Maximum Communication Time	This parameter is ignored in No Wait mode.
Address + 6 to Address + 133	Data Block	The data block contains the command's parameters. The data block begins with a command number in address + 6, which identifies the type of communications function to be performed. Refer to the specific device manual for COMM_REQ command formats.

Status Pointer Memory Type

Status pointer memory type contains a numeric code that specifies the user reference memory type for the status word. The table below shows the code for each reference type:

For this memory type		***Enter this decimal value***
%I	Discrete input table (BIT mode)	70
%Q	Discrete output table (BIT mode)	72
%I	Discrete input table (BYTE mode)	16
%Q	Discrete output table (BYTE mode)	18
%R	Register memory	8
%W	Word memory	196
%AI	Analog input table	10
%AQ	Analog output table	12

Notes:

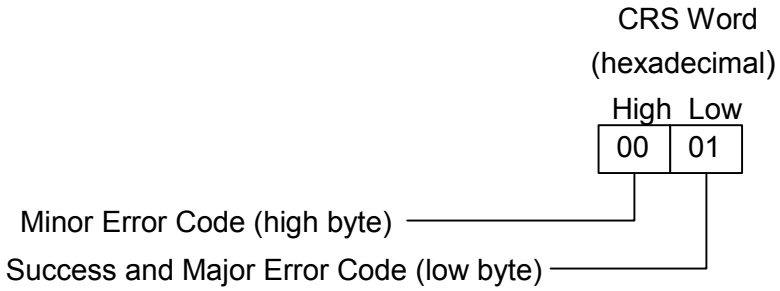
- The value entered determines the mode. For example, if you enter the %I bit mode is 70, then the offset will be viewed as that bit. On the other hand, if the %I value is 16, then the offset will be viewed as that byte.
- The high byte at address + 2 should contain zero.

Operands for COMM_REQ

Parameter	Description	Allowed Operands	Optional
IN	The reference of the first WORD of the command block.	Variables in %R, %P, %L, %AI, %AQ, %W, and symbolic non-discrete variables	No
SYSID	The rack number (most significant byte) and slot number (least significant byte) of the target device (intelligent module). **Note:** For systems that do not have expansion racks, SYSID must be zero for the main rack.	All except flow and variables in %S - %SC	No
TASK	The task ID of the process on the target device	Constants; variables in %R, %P, %L, %AI, %AQ, %W, and symbolic non-discrete variables	No
FT	Function Faulted output. FT is energized if an error is detected processing the COMM_REQ: ■ This is a WAIT mode COMM_REQ and the CPU does not support it ■ The specified target address (SYSID operand) is not present. ■ The specified task (TASK operand) is not valid for the device. ■ The data length is 0. ■ The device's status pointer address (part of the command block) does not exist. This may be due to an incorrect memory type selection, or an address within that memory type that is out of range.	Power flow	Yes

COMM_REQ Status Word

The CRS word consists of two byte values, a major code and a minor code.

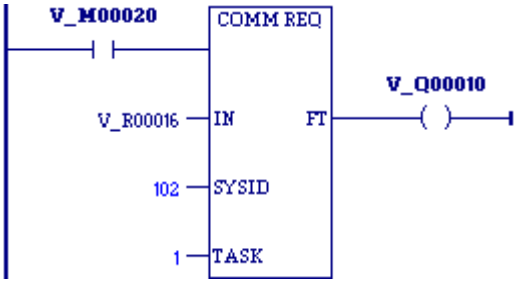


Refer to the specific device manual for CRS major and minor codes used by COMM_REQ commands at that device.

Examples for COMM_REQ

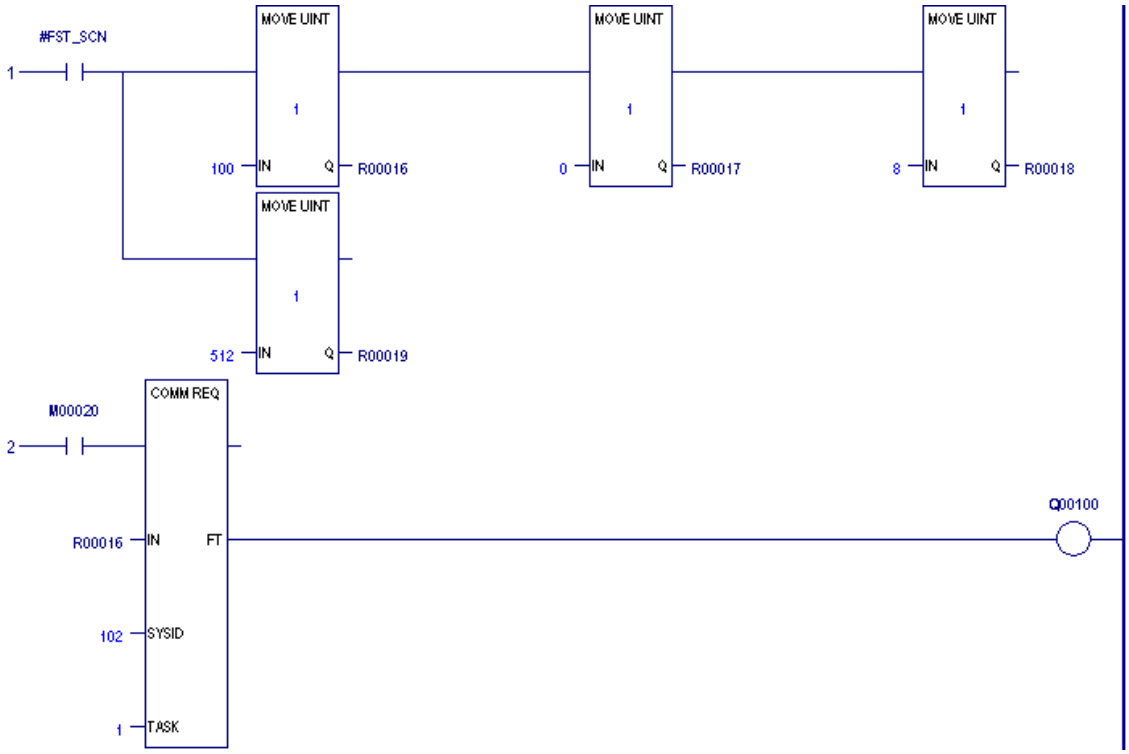
Example 1

When enabling input %M0020 is ON, a command block starting at %R0016 is sent to communications task 1 in the device located at rack 1, slot 2 of the PLC. If an error occurs processing the COMM_REQ, %Q0100 is set.



Example 2

The MOVE function can be used to enter the command block contents for the COMM_REQ described in example 1.



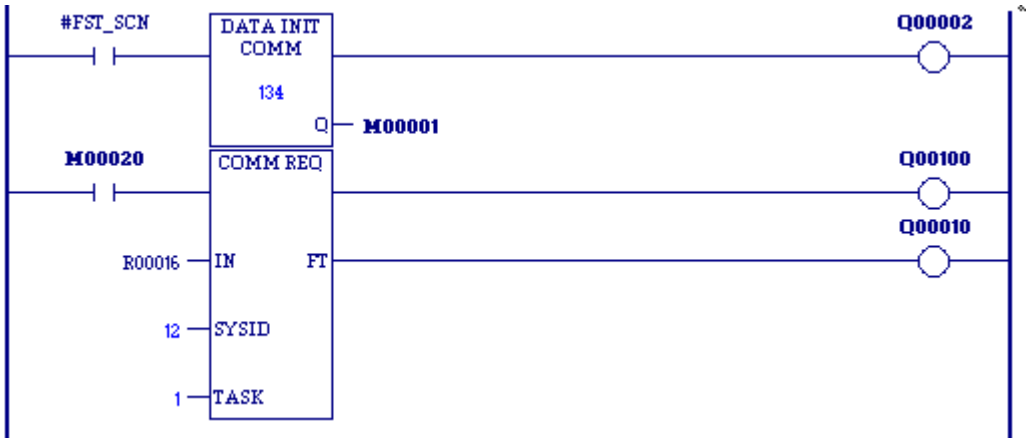
Input IN of the COMM_REQ specifies %R00016 as the beginning reference for the command block. Successive references contain the following:

%R00016	Data Block Length
%R00017	Wait/No Wait Flag
%R00018	Status Pointer Memory Type
%R00019	Status Pointer Offset
%R00020	Idle Timeout Value (Because this parameter is ignored in NO WAIT mode, no value is input).
%R00021	Maximum Communication Time Value (Because this parameter is ignored in NO WAIT mode, no value is input).
%R00022 to end of data	Data Block

MOVE functions supply the following command block data for the COMM_REQ.

- The first MOVE function places the length of the data being communicated in %R00016.
- The second MOVE function places the constant 0 in %R00017. This specifies NO WAIT mode.
- The third MOVE function places the constant 8 in %R00018. This specifies the register table as the location for the status pointer.
- The fourth MOVE function places the constant 512 in reference %R00019. Therefore, the status pointer is located at %R00513.

The programming logic displayed in example 2 can be simplified by replacing the six MOVE functions with one DATA_INIT_COMM function.



Data Initialization

Note: The mnemonics DATA_INIT_ASCII (page 8-90) and DATA_INIT_COMM (page 8-91) operate differently from the other six functions.



Mnemonics:
DATA_INIT_DWORD
DATA_INIT_DWORD
DATA_INIT_INT
DATA_INIT_UINT
DATA_INIT_REAL
DATA_INIT_LREAL
DATA_INIT_WORD

The Data Initialization (DATA_INIT) function copies a block of constant data to a reference range.

When the DATA_INIT instruction is first programmed, the constants are initialized to zeroes. To specify the constant data to copy, double-click the DATA_INIT instruction in the LD editor.

When DATA_INIT receives power flow, it copies the constant data to output Q. DATA_INIT's constant data length (LEN) specifies how much constant data of the function type is copied to consecutive reference addresses starting at output Q. DATA_INIT passes power to the right whenever it receives power.

Notes:

- The output parameter is not included in coil checking.
- If you replace one DATA_INIT instruction (except DATA_INIT_ASCII or DATA_INIT_COMM) with another (except DATA_INIT_ASCII or DATA_INIT_COMM), Logic Developer - PLC attempts to keep the same data. For example, configuring a DATA_INIT_INT with eight rows and then replacing the instruction with a DATA_INIT_DINT would keep the data for the eight rows. Some precision may be lost when replacing a DATA_INIT_ instruction, and a warning message will be displayed when this case is detected.

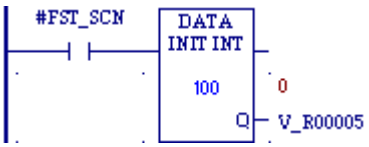
Operands

Note: For each mnemonic, use the corresponding data type for the Q operand. For example, DATA_INIT_DINT requires Q to be a DINT variable.

Parameter	*Description*	*Allowed Operands*	*Optional*
Length	The quantity (default 1) of constant data copied to consecutive reference addresses starting at output Q.	Constants	No
Q	The beginning address of the area to which the data is copied.	All, except %S. SA, SB, and SC are not allowed for REAL, LREAL, INT, and UINT versions.	No

Example

On the first scan (as restricted by the #FST_SCN system variable), 100 words of initial data is copied to %R00005 through %R00104.



Data Initialize ASCII



The Data Initialize ASCII (DATA_INIT_ASCII) function copies a block of constant ASCII text to a reference range.

When DATA_INIT_ASCII is first programmed, the constants are initialized to zeroes. To specify the constant data to copy, double-click the DATA_INIT_ASCII instruction in the LD editor.

When DATA_INIT_ASCII receives power flow, it copies the constant data to output Q. DATA_INIT_ASCII's constant data length (LEN) specifies how many bytes of constant text are copied to consecutive reference addresses starting at output Q. LEN must be an even number. DATA_INIT_ASCII passes power to the right whenever it receives power.

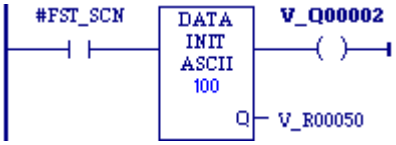
Note: The output parameter is not included in coil checking.

Operands

Parameter	Description	Allowed Operands	Optional
Length	The number (default 1) of bytes of constant text copied to consecutive reference addresses starting at output Q. LEN must be an even number.	Constants	No
Q	The beginning address of the area where the data is copied.	All except %S.	No

Example

On the first scan (as restricted by the #FST_SCN system variable) the decimal equivalent of 100 bytes of ASCII text is copied to %R00050 through %R00149. %Q00002 receives power.



Data Initialize Communications Request

The Data Initialize Communications Request (DATA_INIT_COMM) function initializes a COMM_REQ function with a block of constant data. The IN parameter of the COMM_REQ must correspond with output Q of this DATA_INIT_COMM function.



When DATA_INIT_COMM is first programmed, the constants are initialized to zeroes. To specify the constant data to copy, double-click the DATA_INIT_COMM instruction in the LD editor.

When DATA_INIT_COMM receives power flow, it copies the constant data to output Q. DATA_INIT_COMM's constant data length operand specifies how many words of constant data to copy to consecutive reference addresses starting at output Q. The length should be equal to the size of the COMM_REQ function's entire command block. DATA_INIT_COMM passes power to the right whenever it receives power.

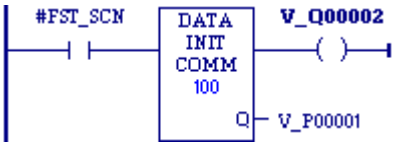
Note: The output parameter is not included in coil checking.

Operands

Parameter	Description	Allowed Operands	Optional
Length	The number of WORDs (default 7) of constant data copied to consecutive reference addresses starting at output Q. Must equal the size of the COMM_REQ function's entire command block, including the header (words 0-5).	Constant	No
Q	The beginning address of the area where the data is copied.	R, W, P, L, AI, AQ, and symbolic non-discrete variables	No

Example

On the first scan (as restricted by the #FST_SCN system variable), a command block consisting of 100 words of data, including the 6 header words, is copied to %P00001 through %P00100. %Q00002 receives power.



Data Initialize DLAN

The Data Initialize DLAN (DATA_INIT_DLAN) function is used with a DLAN Interface module, which is a limited availability, specialty system. If you have a DLAN system, refer to the *DLAN/DLAN+ Interface Module User's Manual,* GFK-0729, for details.

Operands

Parameter	Description	Allowed Operands	Optional
Q	The beginning address of the area where the data is copied.	flow, R, W, P, L, AI, AQ, and symbolic non-discrete variables	No

Move Data



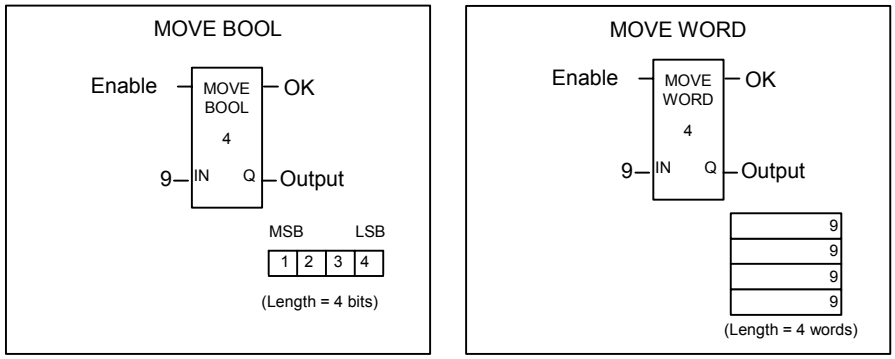
Mnemonics:
MOVE_BOOL
MOVE_DINT
MOVE_DWORD
MOVE_INT
MOVE_REAL
MOVE_UINT
MOVE_WORD

When the MOVE function receives power flow, it copies data as individual bits from one location in PLC memory to another. Because the data is copied in bit format, the new location does not need to be the same data type as the original.

When the MOVE function receives power flow, it copies data from input operand IN to output operand Q as bits. If data is moved from one location in BOOL (discrete) memory to another, for example, from %I memory to %T memory, the transition information associated with the BOOL memory elements is updated to indicate whether or not the MOVE operation caused any BOOL memory elements to change state. Data at the input operand does not change unless there is an overlap in the source and destination.

Note: If an array of BOOL-type data specified in the Q operand does not include all the bits in a byte, the transition bits associated with that byte (which are not in the array) are cleared when the Move function receives power flow. The input IN can be either a variable providing a reference for the data to be moved or a constant. If a constant is specified, then the constant value is placed in the location specified by the output reference. For example, if a constant value of 4 is specified for IN, then 4 is placed in the memory location specified by Q. If the length is greater than 1 and a constant is specified, then the constant is placed in the memory location specified by Q and the locations following, up to the length specified. Do not allow overlapping of IN and Q operands.

The result of the MOVE depends on the data type selected for the function, as shown below. For example, if the constant value 9 is specified for IN and the length is 4, then 9 is placed in the bit memory location specified by Q and the three locations following:



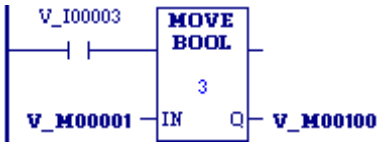
The MOVE function passes power to the right whenever it receives power.

MOVE Data Operands

Parameter	Description	Allowed Operands	Optional
Length (??)	The length of IN; the number of bits, words, or double words to move. If IN is a constant and Q is BOOL, then 1 ≤ Length ≤ 16; otherwise, 1 ≤ Length ≤ 256. 1 ≤ Length ≤ 32,767	Constant	No
IN	The location of the first data item to move. For MOVE_BOOL, any discrete reference may be used. It does not need to be byte-aligned. However 16 bits beginning with the reference address specified are displayed online. If IN is a constant, it is treated as an array of bits. The value of the least significant bit is copied into the memory location specified by Q. If Length is greater than one, the bits are copied in order from the least significant to the most significant into successive memory locations, up to the length specified.	All. S, SA, SB, SC allowed only for WORD, DWORD, BOOL types.	No
Q	The location of the first destination data item. For MOVE_BOOL, any discrete reference may be used. It does not need to be byte-aligned. However 16 bits beginning with the reference address specified are displayed online.	All except %S. Also no %SA, SB, SC except for WORD, DWORD, BOOL types.	

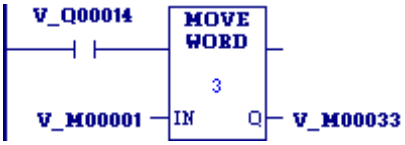
Move Data Example 1

When %I00003 is set, the three bits %M00001, %M00002, and %M00003 are moved to %M00100, %M00101, and %M00102, respectively. Coil %Q00001 is turned on.



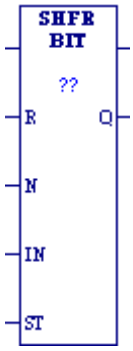
Move Data Example 2

V_M00001 and V_M00033 are both WORD arrays of length 3, for a total of 48 bits in each array. Since PLCs do not recognize arrays, Length has to be set at 3, for the total number of WORDs to be moved. When enabling input V_Q0014 is ON, MOVE_WORD moves 48 bits from the memory location %M00001 to memory location %M00033. Even though the destination overlaps the source for 16 bits, the move is done correctly.



Shift Register

When the Shift Register (SHFR_BIT, SHFR_DWORD, or SHFR_WORD) function receives power and the R operand does not, SHFR shifts one or more data BITs, data DWORDs, or data WORDs from a reference location into a specified area of memory. A contiguous section of memory serves as a shift register. For example, one word might be shifted into an area of memory with a specified length of five words. As a result of this shift, another word of data would be shifted out of the end of the memory area.



Mnemonics:
SHFR_BIT
SHFR_DWORD
SHFR_WORD

Warning

The use of overlapping input and output reference address ranges in multiword functions is not recommended, as it may produce unexpected results.

The reset input (R) takes precedence over the function enable input. When the reset is active, all references beginning at the shift register (ST) up to the length specified, are filled with zeros.

If the function receives power flow and R is not active, each BIT, DWORD, or WORD of the shift register is moved to the next highest reference. The elements shifted out of ST are shifted into Q. The highest reference of IN is shifted into the vacated element starting at ST.

Note: The contents of the shift register are accessible throughout the program because they are overlaid on absolute locations in logic addressable memory.

The function passes power to the right whenever it receives power flow and the R operand does not.

Operands for Shift Register

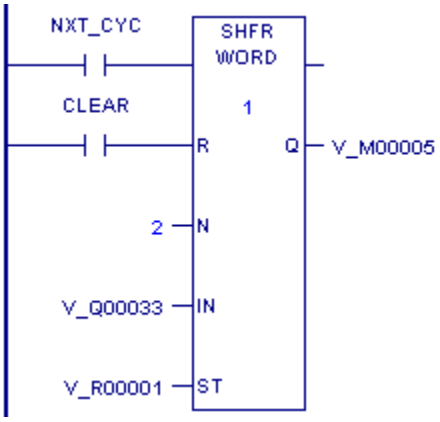
Parameter	Description	Allowed Operands	Optional
Length (??)	The number of data items in the shift register, ST. 1 ≤ Length ≤ 256.		No
R	Reset. When R is ON, the shift register located at ST is filled with zeroes.	Power flow	No
N	The number of data items to shift into ST.	Constants	No
IN	The value to shift into the first data item of ST. **SHFR_BIT:** For %I, %Q, %M and %T memory, any BOOL reference may be used; it does not need to be byte-aligned. However, 1 bit, beginning with the reference address specified, is displayed online.	All	No
ST	The first data item of the shift register. **Note:** For %I, %Q, %M and %T memory, any BOOL reference may be used; it does not need to be byte-aligned. However, 16 bits, beginning with the reference address specified, are displayed online.	All except data flow, constants, S	No

Parameter	Description	Allowed Operands	Optional
Q	The data shifted out of ST. The same number of data items will be shifted into Q as were shifted out of ST. **SHFR_BIT:** For %I, %Q, %M and %T memory, any BOOL reference may be used; it does not need to be byte-aligned. However, 1 bit, beginning with the reference address specified, is displayed online.	All except S	No

Example

SHFR_WORD operates on register memory locations %R0001 through %R0100. When the reset reference CLEAR is active, the Shift Register words are set to zero.

When the NXT_CYC reference is active and CLEAR is not, the two words at the starting address V_Q00033 are shifted into the Shift Register at %R0001. The words shifted out of the Shift Register from %R0100 are stored in output %M0005. Note that, for this example, the length specified for LEN and the amount of data to be shifted (N) are not the same.



Swap

The SWAP function is used to swap two bytes within a word (SWAP WORD) or two words within a double word (SWAP DWORD). The SWAP can be performed over a wide range of memory by specifying a length greater than 1. If that is done, the data in each word or double word within the specified length is swapped.

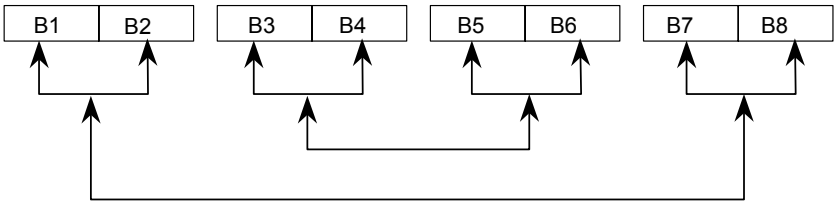


Other mnemonic:
SWAP_WORD

When the SWAP function receives power flow, it swaps the data in reference IN and places the swapped data into output reference Q. The function passes power to the right whenever it receives power.

PACSystems CPUs use the Intel convention for storing word data in bytes. They store the least significant byte of a word in address n and the most significant byte in address n+1. Many VME modules follow the Motorola convention of storing the most significant byte in address n and the least significant byte in address n+1.

The PACSystems CPU assigns byte address 1 to the same storage location regardless of the byte convention used by the other device. However, because of the difference in byte significance, word and multiword data, for example, 16 bit integers (INT, UINT), 32 bit integers (DINT) or floating point (REAL) numbers, must be adjusted when being transferred to or from Motorola-convention modules. In these cases, the two bytes in each word must be swapped, either before or after the transfer. In addition, for multiword data items, the words must be swapped end-for-end on a word basis. For example, a 64-bit real number transferred to the PACSystems CPU from a Motorola-convention module must be byte-swapped and word-reversed, either before or after reading, as shown below:



Character (ASCII) strings or BCD data require no adjustment since the Intel and Motorola conventions for storage of character strings are identical.

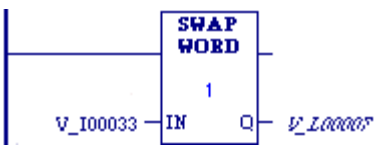
Operands for Swap

The two parameters, IN and Q, must both be the same type, WORD or DWORD.

Parameter	Description	Allowed Operands	Optional
Length (??)	The number of WORDs or DWORDs to operate on. 1 ≤ Length ≤ 256.	Constant	No
IN	Reference for data to be swapped. (must be the same type as Q)	All	No
Q	Reference for swapped data. (must be the same type as IN)	All except S	No

Example for Swap

Two bytes located in bits %I00033 through %I00048 are swapped. The result is stored in %L00007.



Data Table Functions

Function	*Mnemonic*	*Description*
Array Move	ARRAY_MOVE_BOOL ARRAY_MOVE_BYTE ARRAY_MOVE_DINT ARRAY_MOVE_INT ARRAY_MOVE_WORD	Copies a specified number of data elements from a source memory block to a destination memory block. **Note:** The memory blocks do not need to be defined as arrays. You must supply a starting address and the number of contiguous registers to use for the move.
Array Range	ARRAY_RANGE_DINT ARRAY_RANGE_DWORD ARRAY_RANGE_INT ARRAY_RANGE_UINT ARRAY_RANGE_WORD	Determines if a value is between the range specified in two tables
FIFO Read	FIFO_RD_DINT FIFO_RD_DWORD FIFO_RD_INT FIFO_RD_UINT FIFO_RD_WORD	Removes the entry at the bottom of the First In First Out (FIFO) table, and decrements the pointer by one
FIFO Write	FIFO_WRT_DINT FIFO_WRT_DWORD FIFO_WRT_INT FIFO_WRT_UINT FIFO_WRT_WORD	Increments the table pointer and writes data to the bottom of the FIFO table
LIFO Read	LIFO_RD_DINT LIFO_RD_DWORD LIFO_RD_INT LIFO_RD_UINT LIFO_RD_WORD	Removes the entry at the pointer location in the LIFO (Last In First Out) table, and decrements the pointer by one
LIFO Write	LIFO_WRT_DINT LIFO_WRT_DWORD LIFO_WRT_INT LIFO_WRT_UINT LIFO_WRT_WORD	Increments the LIFO table's pointer and writes data to the table
Search	SEARCH_EQ_BYTE SEARCH_EQ_DINT SEARCH_EQ_DWORD SEARCH_EQ_INT SEARCH_EQ_UINT SEARCH_EQ_WORD	Searches for all array values equal to a specified value
	SEARCH_GE_BYTE SEARCH_GE_DINT SEARCH_GE_DWORD SEARCH_GE_INT SEARCH_GE_UINT SEARCH_GE_WORD	Searches for all array values greater than or equal to a specified value
	SEARCH_GT_BYTE SEARCH_GT_DINT SEARCH_GT_DWORD SEARCH_GT_INT SEARCH_GT_UINT SEARCH_GT_WORD	Searches for all array values greater than a specified value

Function	Mnemonic	Description
	SEARCH_LE_BYTE SEARCH_LE_DINT SEARCH_LE_DWORD SEARCH_LE_INT SEARCH_LE_UINT SEARCH_LE_WORD	Searches for all array values less than or equal to a specified value
	SEARCH_LT_BYTE SEARCH_LT_DINT SEARCH_LT_DWORD SEARCH_LT_INT SEARCH_LT_UINT SEARCH_LT_WORD	Searches for all array values less than a specified value
	SEARCH_NE_BYTE SEARCH_NE_DINT SEARCH_NE_DWORD SEARCH_NE_INT SEARCH_NE_UINT SEARCH_NE_WORD	Searches for all array values not equal to a specified value
Sort	SORT_INT SORT_UINT SORT_WORD	Sorts a memory block in ascending order
Table Read	TBL_RD_DINT TBL_RD_DWORD TBL_RD_INT TBL_RD_UINT TBL_RD_WORD	Copies a value from a specified table location to an output reference
Table Write	TBL_WRT_DINT TBL_WRT_DWORD TBL_WRT_INT TBL_WRT_UINT TBL_WRT_WORD	Copies a value from an input reference to a specified table location

Array Move



Mnemonics:

ARRAY_MOVE_BOOL
ARRAY_MOVE_BYTE
ARRAY_MOVE_DINT
ARRAY_MOVE_DWORD
ARRAY_MOVE_INT
ARRAY_MOVE_UINT
ARRAY_MOVE_WORD

When the Array Move function receives power flow, it copies a specified number of elements from a source memory block to a destination memory block. Starting at the indexed location (SR+SNX-1) of the input memory block, it copies N elements to the output memory block, starting at the indexed location (DS+DNX-1) of the output memory block.

Note: For ARRAY_MOVE_BOOL, when 16-bit registers are selected for the operands of the source memory block and/or destination memory block starting address, the least significant bit of the specified 16-bit register is the first bit of the memory block. The value displayed contains 16 bits, regardless of the length of the memory block.

The indices in an Array Move instruction are 1-based. In using an Array Move, no element outside either the source or destination memory blocks (as specified by their starting address and length) may be referenced.

The function passes power flow unless one of the following conditions occurs:

- It receives no power flow.
- (N + SNX - 1) is greater than Length.
- (N + DNX - 1) is greater than Length.

Note: For each mnemonic, use the corresponding data type for the SR and DS operands. For example, ARRAY_MOVE_BYTE requires SR and DS to be BYTE variables.

Operands for Array Move

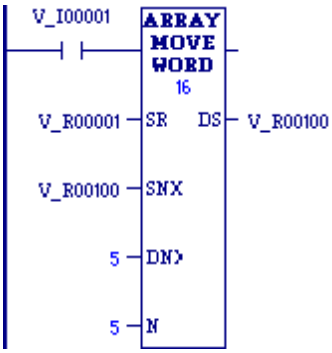
Parameter	Description	Allowed Operands	Optional
Length (??)	The length of each memory block (source and destination); the number of elements in each memory block. 1 ≤ Length ≤ 32,767.	Constant	No
SR (must be the same data type as DS)	The starting address of the source memory block. **Note:** For an Array Move with the data type BOOL, any reference may be used; it does not need to be byte-aligned. Sixteen bits, beginning with the reference address specified, are displayed online.	All except constants. %S - %SC allowed only for BYTE, WORD, DWORD types.	No
SNX	The index of the source memory block	All except variables in %S - %SC.	No
DNX	The index of the destination memory block	All except variables in %S - %SC.	No
N	Count indicator	All except variables in %S - %SC	No
DS (must be the same data type as SR)	The starting address of the destination memory block. **Note:** For an Array Move with the data type BOOL, any reference may be used; it does not need to be byte-aligned. Sixteen bits, beginning with the reference address specified, are displayed online.	All, except S and constants. %SA - %SC allowed only for BYTE, WORD, DWORD types	No

Examples for Array Move

Example 1

To define the input memory block %R0001 - %R0016 and the output memory block %R0100 - %R0115, SR is set as %R0001, DS is set as %R0100, and Length is set to 16.

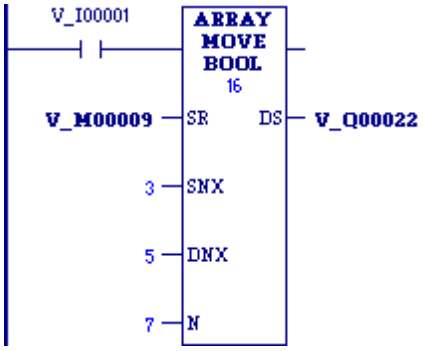
To copy the five registers %R0003 - %R0007 to the registers %R0104 - %R0108, N is set to 5, SNX=%R0100 is set to 3 (to designate the third register, %R0003, of the block starting at %R0001), and DNX is set to 5 (to designate the fifth register, %R0104, of the block starting at %R0100).



Example 2

Using bit memory blocks, the input block starts at SR=%M0009, the output block starts at %Q0022, and the length of both blocks is 16 one-bit registers (Length=16).

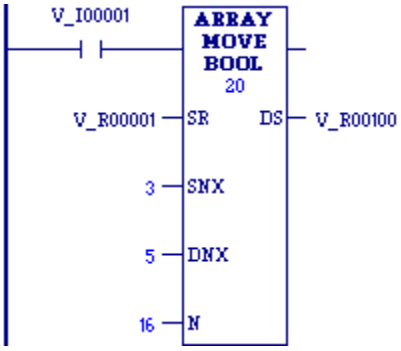
To copy the seven registers %M0011 - %M0017 to %Q0026 - %Q0032, N is set to 7, SNX is set to 3 (to designate the third register, %M0011, of the block starting at %M0009), and DNX is set to 5 (to designate the fifth register, %Q0026, of the block starting at %Q0022).



Example 3

Sixteen (=N) bits that are not byte-aligned are moved from the two 16-bit registers that start at %R00001 (SR) to the two 16-bit registers that begin at %R00100 (DS). For the purposes of this Boolean move, Length is set to 20, because the other 12 bits in either memory block are not considered.

By setting SNX to 3, N to 16, and DNX to 5, the third (SNX) least significant bit of %R0001 through the second least significant bit of %R0002 (for a total of 16 bits=N) are written into the fifth (DNX) least significant bit of %R0100 through the fourth least significant bit of %R0101 (for the same total of 16 bits).



Array Range



Mnemonics:
ARRAY_RANGE_DINT
ARRAY_RANGE_DWORD
ARRAY_RANGE_INT
ARRAY_RANGE_UINT
ARRAY_RANGE_WORD

The ARRAY_RANGE function compares a single input value against two arrays of delimiters that specify an upper and lower bound to determine if the input value falls within the range specified by the delimiters. The output is an array of bits that is set ON (1) when the input value is greater than or equal to the lower limit and less than or equal to the upper limit. The output is set OFF (0) when the input is outside this range or when the range is invalid, as when the lower limit exceeds the upper limit.

The ARRAY_RANGE function compares a single input value against two arrays of delimiters that specify an upper and lower bound to determine if the input value falls within the range specified by the delimiters. The output is an array of bits that is set ON (1) when the input value is greater than or equal to the lower limit and less than or equal to the upper limit. The output is set OFF (0) when the input is outside this range or when the range is invalid, as when the lower limit exceeds the upper limit.

When ARRAY_RANGE receives power, it compares the value in input parameter IN against each range specified by the array element values of LL and UL. Output Q sets a bit ON (1) for each corresponding array element where the value of IN is greater than or equal to the value of LL and is less than or equal to the value of UL. Output Q sets a bit OFF (0) for each corresponding array element where the value of IN is not within this range or when the range is invalid, as when the value of LL exceeds the value of UL. If the operation is successful, ARRAY_RANGE passes power flow to the right.

Operands for Array Range

Notes:

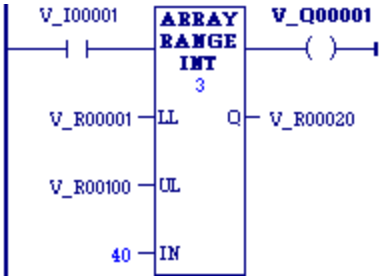
- For each mnemonic, use the corresponding data type for the LL, UL, and Q operands. For example, ARRAY_RANGE_DINT requires LL, UL, and Q to be DINT variables.
- Q is not aligned. It is displayed in bit format. It displays either a 1 (ON) or a 0 (OFF) for the first array element. For BOOL references, it represents the reference displayed. For other references, it represents the low order bit of the reference displayed.

Parameter	Description	Allowed Operands	Operands Optional
Length (??)	The number of elements in each array.	Constant	No
LL	The lower limit of the range	All except constants and %S - %SC for INT, DINT.	No
UL	The upper limit of the range	All except constants and %S - %SC for INT, DINT.	No
IN	The value to compare against each range specified by LL and UL	All except constants and %S - %SC for INT, DINT.	No
Q	Energized when the value in IN is within the range specified by LL and UL, inclusive.	All except S	No

Examples for Array Range

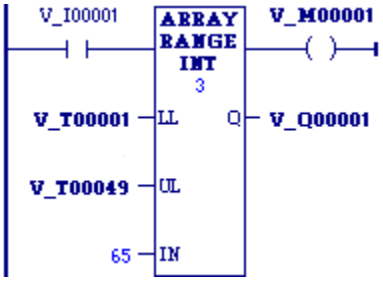
Example 1

The lower limit (LL) values of %R00001 through %R00008 are 1, 20, 30, 100, 25, 50, 10, and 200. The upper limit (UL) values of %R00100 through %R00108 are 40, 50, 150, 2, 45, 90, 250, and 47. The resulting Q values will be placed in the first 8 bits of %R00200. The bit values low order to high are: 1, 1, 1, 0, 1, 0, 1, and 0. The bit value displayed will be set ON (1) for the low order bit of %R00200. The ok output will be set ON (1).



Example 2

The lower limit (LL) array contains %T00001 through %T00016, %T00017 through %T00032, and %T00033 through %T00048. The lower limit values are 100, 65, and 1. The upper limit (UL) values are 29, 165, and 2. The resulting Q values of 0, 1, and 0 will be placed in %Q00001 through %Q00003. The bit value displayed will be 0 (OFF), representing the value of %Q00001. The power output will be set ON (1).



FIFO Read



Mnemonics:

FIFO_RD_DINT
FIFO_RD_DWORD
FIFO_RD_INT
FIFO_RD_UINT
FIFO_RD_WORD

The First-In-First-Out (FIFO) Read (FIFO_RD) function moves data out of tables. Values are always moved out of the bottom of the table. If the pointer reaches the last location and the table becomes full, FIFO_RD must be used to remove the entry at the pointer location and decrement the pointer by one. FIFO_RD is used in conjunction with the FIFO_WRT function, which increments the pointer and writes entries into the table.

1. FIFO_RD copies the top location (entry 0) of the table to output parameter Q. Additional program logic must then be used to place the data in the input reference.
2. The remaining items in the table are copied to a lower numbered position in the table.
3. FIFO_RD decrements the pointer by one.
4. Steps 1, 2, and 3 are repeated each time FIFO_RD is executed, until the table is empty (PTR = 0).

The pointer does not wrap around when the table is full.

When FIFO_RD receives power flow, the data at the first location of the table is copied to output Q. Next, each item in the table is moved down to the next lower location. This begins with item 2 in the table, which is moved into position 1. Finally, the pointer is decremented. If this causes the pointer location to become 0, the output EM is set ON, i.e., EM indicates whether or not the table is empty.

FIFO_RD passes power to the right if the pointer is greater than zero and less than the value specified for LEN.

Note: A FIFO table is a queue. A LIFO table is a stack.

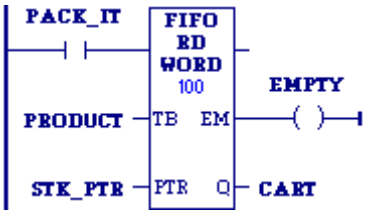
Operands for FIFO Read

Note: For each mnemonic, use the corresponding data type for the TB and Q operands. For example, FIFO_RD_DINT requires TB and Q to be DINT variables.

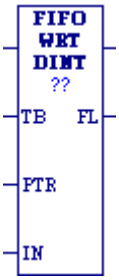
Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	1 ≤ Length ≤ 32,767.	Constants	No
TB (must be the same type as Q)	The elements in the FIFO table	All except constants	No
PTR	Pointer. Index of the last element of the FIFO table.	All except constants, data flow, and variables in %S -%SC	No
EM	Energized when the last element of the table is read	Flow	No
Q (must be the same type as TB)	The element read from the FIFO table	All except constants, S; SA, SB, SC allowed only for WORD, DWORD	No

Example for FIFO Read

PRODUCT is a FIFO table with 100 word-sized elements. When the enabling input PACK_IT is ON, the PRODUCT data item in the table location pointed to by STK_PTR is copied to the reference location specified in CART. This table location pointed to would be the bottom, or oldest data item in the table. The number in STK_PTR is then decremented. A copy of the oldest data item in the PRODUCT table is left behind in each table location as the current data is copied out during successive PACK_IT triggers. Output node EM passes power when the PTR = 0, firing the coil EMPTY. No further data from the PRODUCT table can be read without first copying data in using the FIFO_WRT function.



FIFO Write



Mnemonics:

FIFO_WRT_DINT
FIFO_WRT_DWORD
FIFO_WRT_INT
FIFO_WRT_UINT
FIFO_WRT_WORD

The First-In-First-Out (FIFO) Write (FIFO_WRT) function moves data into tables. The function increments the table pointer by one and adds an entry at the new pointer location in a FIFO table. Values are always moved in at the bottom of the table. If the pointer reaches the last location and the table becomes full, FIFO_WRT can add no further values. The FIFO_RD function must then be used to remove the entry at the pointer location and decrement the pointer by one.

1. FIFO_WRT increments the pointer by one.
2. FIFO_WRT copies data from input parameter IN to the position in the table indicated by the pointer. (It writes over any value currently at that location.) Additional program logic must then be used to place the data in the input reference.
3. Steps 1 and 2 are repeated each time FIFO_WRT is executed, until the table is full (PTR=0).

The pointer does not wrap around when the table is full.

When FIFO_WRT receives power flow, the pointer is incremented by 1. Then, input data is written into the table at the pointer location. If the pointer was already at the last location in the table, no data is written and FIFO_WRT does not pass power to the right. The pointer always indicates the last item entered into the table. If the table becomes full, it is not possible to add more entries to it.

FIFO_WRT passes power to the right after a successful execution (PTR < LEN).

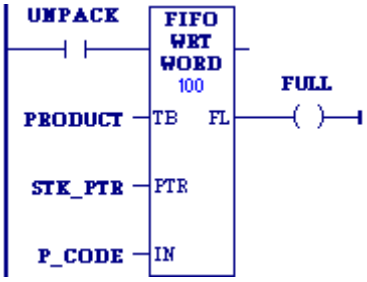
Operands for FIFO Write

Note: For each mnemonic, use the corresponding data type for the TB and IN operands. For example, FIFO_WRT_DINT requires TB and IN to be DINT variables.

Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	1 ≤ Length ≤ 32,767.	Constants	No
TB (must be the same data type as IN)	The elements in the FIFO table	All except constants, data flow, and S. SA - SC allowed only for WORD, DWORD types	No
PTR	Pointer. Index of the last element of the FIFO table.	All except constants, data flow, S - SC.	No
IN (must be the same data type as TB)	The element to write to the FIFO table	All. S – SC allowed only for WORD, DWORD types.	No
FL	Energized when IN is written to the last element of the table	Power flow	No

Example for FIFO Write

PRODUCT is a FIFO table with 100 word-sized elements. When the enabling input UNPACK is ON, a data item from P_CODE is copied to the table location pointed to by the value in STK_PTR. Output node FL passes power when PTR = LEN, firing the FULL coil. No further data from P_CODE can be added to the table without first copying data out, using the FIFO_RD function.



LIFO Read



Mnemonics:

LIFO_RD_DINT
LIFO_RD_DWORD
LIFO_RD_INT
LIFO_RD_UINT
LIFO_RD_WORD

The Last-In-First-Out (LIFO) Read (LIFO_RD) function moves data out of tables. Values are always moved out of the top of the table. If the pointer reaches the last location and the table becomes full, LIFO_RD must be used to remove the entry at the pointer location and decrement the pointer by one. LIFO_RD is used in conjunction with the LIFO_WRT function, which increments the pointer and writes entries into the table.

1. LIFO_RD copies data indicated by the pointer to output parameter Q. Additional program logic must then be used to place the data in the input reference.
2. LIFO_RD decrements the pointer by one.
3. Steps 1 and 2 are repeated each time the instruction is executed, until the table is empty (PTR = LEN).

The pointer does not wrap around when the table is full.

When LIFO_RD receives power flow, the data at the pointer location is copied to output Q, then the pointer is decremented. If this causes the pointer location to become 0, the output EM is set ON, i.e., EM indicates whether or not the table is empty. If the table is empty when LIFO_RD receives power flow, no read occurs. The pointer always indicates the last item entered into the table.

LIFO_RD passes power to the right if the pointer was in range for an element to be read.

Note: A LIFO table is a stack. A FIFO table is a queue.

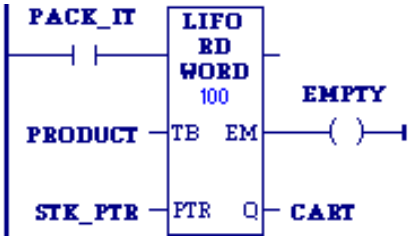
Operands for LIFO Read

Note: For each mnemonic, use the corresponding data type for the TB and Q operands. For example, LIFO_RD_DINT requires TB and Q to be DINT variables.

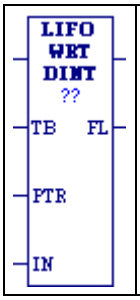
Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	1 ≤ Length ≤ 32,767.	Constant	No
TB (must be the same type as Q)	The elements in the table	All except constants	No
PTR	Pointer. Index of the next element to read.	All except constants, S - SC, and data flow	No
EM	Energized when the last element of the table is read	Power flow	No
Q (must be the same type as TB)	The element read from the table	All except constants and S. SA, SB, SC allowed only for WORD, DWORD.	No

Example for LIFO Read

PRODUCT is a LIFO table with 100 word-sized elements. When the enabling input PACK_IT is ON, the data item at the top of the table is copied into the reference indicated by the nickname CART. The reference identified by STK_PTR contains the table pointer. Output coil EMPTY indicates when the table is empty.



LIFO Write



Mnemonics:
LIFO_WRT_DINT
LIFO_WRT_DWORD
LIFO_WRT_INT
LIFO_WRT_UINT
LIFO_WRT_WORD

The Last-In-First-Out (LIFO) Write (LIFO_WRT) function increments the table pointer by one and then adds an entry at the new pointer location in a table. Values are always moved in at the top of the table. If the pointer reaches the last location and the table becomes full, LIFO_WRT cannot add further values. LIFO_RD must then be used to remove the entry at the pointer location and decrement the pointer by one.

1. LIFO_WRT increments the table pointer by one.
2. LIFO_WRT copies data from input parameter IN to the position in the table indicated by the pointer. (It writes over any value currently at that location.) Additional program logic must then be used to place the data in the input reference.
3. Steps 1 and 2 are repeated each time LIFO_WRT is executed, until the table is full (PTR=LEN).

The pointer does not wrap around when the table is full.

When LIFO_WRT receives power flow, the pointer increments by 1; then the new data is written at the pointer location. If the pointer was already at the last location in the table, no data is written and LIFO_WRT does not pass power to the right. The pointer always indicates the last item entered into the table. If the table is full, it is not possible to add more entries to it.

LIFO_WRT passes power to the right after a successful execution (PTR < LEN).

Note: A LIFO table is a stack. A FIFO table is a queue.

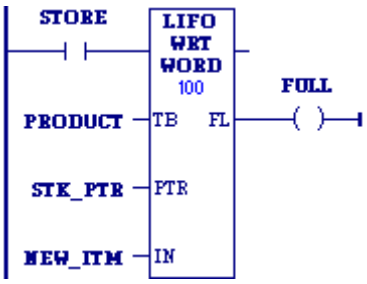
Operands for LIFO Write

Note: For each mnemonic, use the corresponding data type for the TB and IN operands. For example, LIFO_WRT_DINT requires TB and Q to be DINT variables.

Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	1 ≤ Length ≤ 32,767.	Constants	No
TB (must be the same type as IN)	The elements in the table	All except constants, S, data flow. SA - SC allowed only for WORD, DWORD.	No
PTR	Pointer. Index of the next element to write.	All except constants, S - SC, and data flow	No
IN (must be the same type as TB)	The element to write to the table	All. S – SC allowed only for WORD, DWORD	No
FL	Energized when IN is written to the last element of the table	All	No

Example for LIFO Write

PRODUCT is a LIFO table with 100 word-sized elements. When the enabling input STORE is ON, a data item from NEW_ITEM is copied to the table location pointed to by the value in STK_PTR. Output FL passes power when PTR = LEN, firing the FULL coil. No further data from NEW_ITEM can be added to the table without first copying data out, using the LIFO_RD function.



Search



When the Search function receives power, it searches the specified memory block for a value that satisfies the search criteria. For example, SEARCH_GE_DWORD searches for a DWORD that is greater than or equal to the specified value (the IN operand).

Search can evaluate six different relationships for six data types, for a total of thirty-six mnemonics.

Search Relationships:

SEARCH_EQ_...	searches for a value of the specified data type ***equal*** to the IN operand.
SEARCH_GE_...	searches for a value of the specified data type ***greater than or equal*** to IN.
SEARCH_GT_...	searches for a value of the specified data type ***greater than*** IN.
SEARCH_LE_...	searches for a value of the specified data type ***less than or equal*** to IN.
SEARCH_LT_...	searches for a value of the specified data type ***less than*** IN.
SEARCH_NE_...	searches for a value of the specified data type that is ***not equal*** to IN.

Data types:

BYTE, DINT, DWORD, INT, UINT, WORD

Searching begins at AR+INX, where AR is the starting address and INX is the index value into the memory block. The search continues either until a register that satisfies the search criteria is found or until the end of the memory block is reached.

- If a register is found, the Found Indication (FD) is set ON and the Output Index (ONX) is set to the relative position of this register within the block.
- If no register is found before the end of the block is reached, the Found Indication (FD) is set OFF and the Output Index (ONX) is set to zero.

The input index (INX) is zero-based, that is, 0 the means first reference, whereas the output index (ONX) is one-based, that is, 1 means the first reference.

The valid values for INX are 0 to (Length - 1). The valid values for ONX are 1 to Length.

INX should be set to zero to begin searching at the memory block's first register. This value increments by one at the time of execution. If the value of input INX is out-of-range, (< 0 or > Length-1), INX is set to the default value of zero.

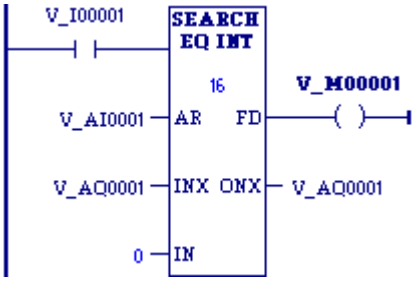
SEARCH passes power flow to the right when it performs without error. If INX is out of range, SEARCH does not pass power flow to the right.

Operands for the Search Function

Note: For each mnemonic, use the corresponding data type for the AR and IN operands. For example, SEARCH_EQ_BYTE requires AR and IN to be BYTE variables.

Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	The number of registers starting at AR that make up the memory block to search. 1 ≤ Length ≤ 32,767 8-bit or 16-bit registers.	Constants	No
AR (must be the same type as IN)	The starting address of the memory block to search; the address of the first register in the memory block.	All except constants	No
INX	The zero-based index into the memory block at which to begin the search. Zero points to the first reference. Valid range: 0 ≤ INX ≤ (Length-1). If INX is out of range, it is set to the default value of 0.	All except constants	No
IN (must be the same type as AR)	The value that the search is based on. For example: SEARCH_GT_DINT searches for a DINT value that is greater than IN. SEARCH_NE_UINT searches for a UINT value that is not equal to IN. SEARCH_GE_WORD searches for a WORD value that is greater than or equal to IN.	All	No
ONX	The one-based position within the memory block of the search target. A value of 1 points to the first reference. Valid range: 1 ≤ ONX ≤ Length	data flow, I, Q, M, T, G, R, P, L, AI, AQ	No
FD	Found indicator. This power flow indicator is energized when a register that satisfies the search criteria is found and the function was successful.	Power flow	No

Example for the Search Function



To search the memory block %AI00001 - %AI00016, AR is set as %AI00001 and Length is set as 16. The values of the 16 registers are 100, 20, 0, 5, 90, 200, 0, 79, 102, 80, 24, 34, 987, 8, 0, and 500. Initially, the search index into AR, %AQ0001, is 5. When power flow input is ON, each scan searches the memory block looking for a match to the IN value of 0. The first scan starts searching at %AI00006 and finds a match at %AI00007, so FD turns ON and %AQ00001 becomes 7. The second scan starts searching at %AI00008 and finds a match at %AI00015, so FD remains ON and %AQ0001 becomes 15. The next scan starts at %AI00016. Since the end of the memory block is reached without a match, FD is set OFF and %AQ0001 is set to zero. The next scan starts searching at the beginning of the memory block.

Sort



Mnemonics:
SORT_INT
SORT_UINT
SORT_WORD

When it receives power flow, the SORT function sorts the elements of the memory block 'IN' in ascending order. The output memory block Q contains integers that give the index that the sorted elements had in the original memory block or list. Q is exactly the same size as IN. It also has a specification (LEN) of the number of elements to be sorted.

SORT operates on memory blocks of no more than 64 elements. When EN is ON, all of the elements of IN are sorted into ascending order, based on their data type. The array Q is also created, giving the original position that each sorted element held in the unsorted array. OK is always set ON.

Notes: The SORT function is executed each scan it is enabled.

Do not use the SORT function in a timed or triggered input program block.

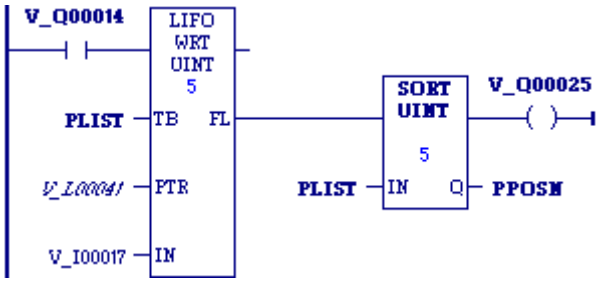
Operands

Note: For each mnemonic, use the corresponding data type for the IN and Q operands. For example, SORT_INT requires IN and Q to be INT variables.

Parameter	*Description*	*Allowed Operands*	*Optional*
Length (??)	The number (1—64) of elements that make up the memory block to sort.	Constants	No
IN	The memory block that contains the elements to sort. After the sort, IN contains the elements in the sorted order.	All except data flow, S, constants. SA – SC valid only for WORD type	No
Q (must be the same type as IN)	An array of indexes that gives the position of the sorted elements in the original memory block	All except S - SC and constants	No

Example

New part numbers (%I00017 - %I00032) are pushed onto a parts array PLIST every time %Q00014 is ON. When the array is filled, it is sorted and the output %Q00025 is turned on. The array PPOSN then contains the original position that the now-sorted elements held before the sort was done on PLIST.



If PLIST was an array of five elements and contained the values 25, 67, 12, 35, 14 before the sort, then after the sort it would contain the values 12, 14, 25, 35, 67. PPOSN would contain the values 3, 5, 1, 4, 2.

Table Read

The Table Read (TBL_RD) function sequentially reads values in a table. When the pointer reaches the end of the table, it wraps around to the beginning of the table. (TBL_RD is like FIFO_RD with a wrap-around.)



Mnemonics:
TBL_RD_DINT
TBL_RD_DWORD
TBL_RD_INT
TBL_RD_UINT
TBL_RD_WORD

When TBL_RD receives power flow:

1. TBL_RD increments the pointer by one.
2. TBL_RD copies data indicated by the pointer to output parameter Q. Additional program logic must then be used to capture the data from the output reference.
3. Steps 1 and 2 are repeated each time the instruction is executed, until the end of the table is reached (PTR=the length specified in Length). When the end of the table is reached, the pointer wraps around to the beginning of the table.

When TBL_RD receives power flow, the pointer (PTR) increments by one. If this new pointer location is the last item in the table, the output EM is set ON. The next time TBL_RD executes, PTR is automatically set back to 1. After PTR is incremented, the content at the new pointer location is copied to output Q.

TBL_RD always passes power to the right when it receives power.

Note: The TBL_RD and TBL_WRT functions can operate on the same or different tables. By specifying a different reference for the pointer, these functions can access the same data table at different locations or at different rates.

Operands

Note: For each mnemonic, use the corresponding data type for the TB and Q operands. For example, TBL_RD_DINT requires TB and Q to be DINT variables.

Parameter	*Description*	*Allowed Operands*	*Optional*
Length	1 ≤ Length ≤ 32,767	Constants	No
TB (must be the same type as Q)	The elements in the table	All except constants	No
PTR	Pointer. Index of the next element.	All except data flow, S - SC, constants	No
EM	Energized when the last element of the table is read	Power flow	No
Q (must be the same type as TB)	The element read from the table	All except constants, S. SA, SB, SC allowed only for WORD, DWORD	No

Example

WIDGETS is a table with 20 integer elements. When the enabling input %M00346 is ON, the pointer increments and the contents of the next element of the table are copied into ITEM_CT. %L00001 functions as the pointer into the data table. %M01001 is used to signal when all items of the data table have been accessed.

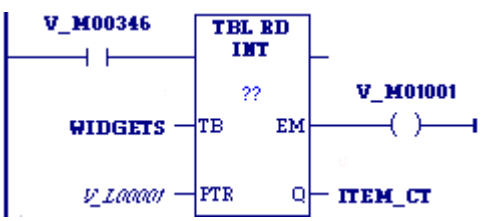
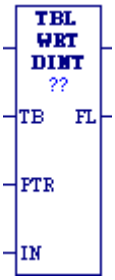


Table Write



Mnemonics:
TBL_WRT_DINT
TBL_WRT_DWORD
TBL_WRT_INT
TBL_WRT_UINT
TBL_WRT_WORD

The Table Write (TBL_WRT) function sequentially updates values in a table that never becomes full. When the pointer (PTR) reaches the end of the table, it automatically returns to the beginning of the table.

1. TBL_WRT increments the pointer by one.
2. TBL_WRT copies data from input parameter IN to the position in the table indicated by the pointer. (It writes over any value currently at that location.) Additional program logic must then be used to place the data in the input reference.
3. Steps 1 and 2 are repeated each time the instruction is executed, until the table is full (PTR=LEN).

When the table is full, the pointer wraps around to the beginning of the table.

Note: The TBL_WRT and TBL_RD functions can operate on the same or different tables. By specifying a different reference for the pointer, these functions can access the same data table at different locations or at different rates.

When TBL_WRT receives power flow, the pointer (PTR) increments by 1. If this new pointer location is the last item in the table, the output FL is set to ON. The next time TBL_WRT executes, PTR is automatically set back to 1. After incrementing PTR, TBL_WRT writes the content of the input reference to the current pointer location, overwriting data already stored there.

TBL_WRT always passes power to the right when it receives power.

Note: TBL_WRT is like FIFO_WRT with a wrap-around.

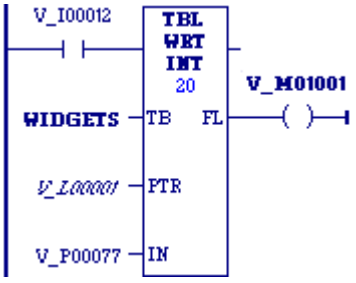
Operands

Note: For each mnemonic, use the corresponding data type for the TB and IN operands. For example, TBL_WRT_DINT requires TB and IN to be DINT variables.

Parameter	*Description*	*Allowed Operands*	*Optional*
Length	1 ≤ Length ≤ 32,767.	Constants	No
TB (must be the same data type as IN)	The elements in the table	All except S, constants, data flow. SA – SC allowed only for WORD, DWORD	No
PTR	Pointer. Index of the next element.	All except constants, data flow, %S - %SC	No
IN (must be the same data type as TB)	The element to write to the table	All. %S - %SC allowed only for WORD, DWORD	No
FL	Energized when IN is written to the last element of the table	Power flow	No

Table Write Example

WIDGETS is a table with 20 integer elements. When enabling input %I00012 is ON, the pointer increments the contents of %P00077 are written into the table at pointer location. %L00001 functions as the pointer into data table.



the and the the

Math Functions

Your program may need to include logic to convert data to a different type before using a Math or Numerical function. The description of each function includes information about appropriate data types. The “Conversion Functions” section on page 8-58 explains how to convert data to a different type.

Function	*Mnemonics*	*Description*
Absolute Value	ABS_DINT, ABS_INT, ABS_REAL, ABS_LREAL	Finds the absolute value of a double- precision integer (DINT), signed single-precision integer (INT), or floating-point (REAL or LREAL) value. The mnemonic specifies the value's data type.
Add	ADD_DINT, ADD_INT, ADD_REAL, ADD_LREAL, ADD_UINT	Addition. Adds two numbers.
Divide*	DIV_DINT, DIV_INT, DIV_MIXED, DIV_REAL, DIV_LREAL, DIV_UINT	Division. Divides one number by another and outputs the quotient. **Note:** Take care to avoid overflow conditions when performing divisions.
Modulus	MOD_DINT, MOD_INT, MOD_UINT	Modulo Division. Divides one number by another and outputs the remainder.
Multiply*	MUL_DINT, MUL_INT, MUL_MIXED, MUL_REAL, MUL_LREAL, MUL_UINT	Multiplication. Multiplies two numbers. **Note:** Take care to avoid overflow conditions when performing multiplications.
Scale	SCALE	Scales an input parameter and places the result in an output location.
Subtract	SUB_DINT, SUB_INT, SUB_REAL, SUB_LREAL, SUB_UINT	Subtraction. Subtracts one number from another.

* To avoid overflows when multiplying or dividing 16-bit numbers, use the conversion functions described on page 8-58 to convert the numbers to a 32-bit format.

Overflow

When an operation results in overflow, there is no power flow.

If an operation on signed operands (INT, DINT, REAL) results in overflow, the output reference is set to its largest possible value for the data type. For signed numbers, the sign is set to show the direction of the overflow. If signed or double precision integers are used, the sign of the result for DIV and MUL functions depends on the signs of I1 and I2.

Maximum Values	MAXINT16	Maximum signed 16-bit	7FFF hex	32,767
	MAXUINT16	Maximum unsigned 16-bit	FFFF hex	65,535
	MAXINT32	Maximum signed 32-bit	7FFFFFFF hex	2,147,483,647
Minimum Values	MININT16	Minimum signed 16-bit	8000 hex	–32,768
	MININT32	Minimum signed 32-bit	80000000 hex	–2,147,483,648

If an operation on unsigned operands (UINT) results in overflow or underflow, the output value wraps around. For example the ADD_UINT operation, 65535+16, yields a result of 15.

Absolute Value



Mnemonics:

ABS_DINT
ABS_INT
ABS_REAL
ABS_LREAL

When the function receives power flow, it places the absolute value of input IN into output Q.

The function outputs power flow, unless one of the following conditions occurs:

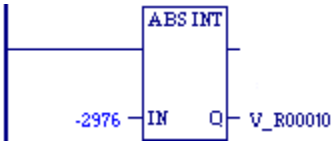
- For INT type, IN is –32,768.
- For DINT type, IN is –2,147,483,648.
- For REAL or LREAL type, IN is NaN (Not a Number).

Operands

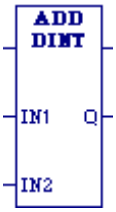
Parameter	Description	Allowed Operands	Optional
IN (must be same type as Q)	The value to process.	All except S, SA, SB, SC	No
Q (must be same type as IN)	The absolute value of IN.	All except S, SA, SB, SC and constant	No

Example

The absolute value of –2,976, which is 2,976, is placed in %R00010:



Add



Mnemonics:

ADD_DINT
ADD_INT
ADD_REAL
ADD_LREAL
ADD_UINT

When the ADD function receives power flow, it adds the two operands IN1 and IN2 of the same data type and stores the sum in the output variable assigned to Q, also of the same data type.

The power flow output is energized when ADD is performed, unless an invalid operation or overflow occurs. (For more information, see “Overflow” on page 8-115.)

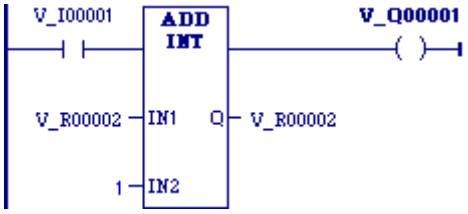
Mnemonic	*Operation*	*Displays as*
ADD_INT	Q(16 bit) = IN1(16 bit) + IN2(16 bit)	base 10 number with sign, up to 5 digits long
ADD_DINT	Q(32 bit) = IN1(32 bit) + IN2(32 bit)	base 10 number with sign, up to 10 digits long
ADD_REAL	Q(32 bit) = IN1(32 bit) + IN2(32 bit)	base 10 number, sign and decimals, up to 8 digits long (excluding the decimals)
ADD_LREAL	Q(64 bit) = IN1(64 bit) + IN2(64 bit)	base 10 number, sign and decimals, up to 17 digits long (excluding the decimals)
ADD_UINT	Q(16 bit) = IN1(16 bit) + IN2(16 bit)	base 10 number, unsigned, up to 5 digits long

Operands of the ADD Function

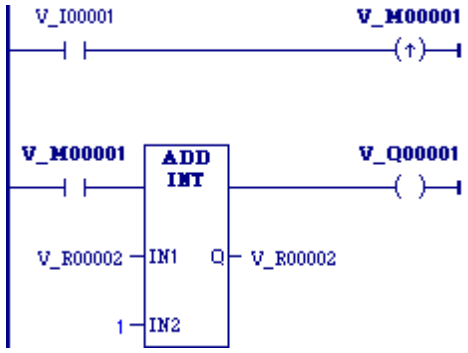
Operand	Description	Allowed Operands	Optional
IN1	The value to the left of the plus sign (+) in the equation IN1+IN2=Q.	All except S, SA, SB, SC	No
IN2	The value to the right of the plus sign (+) in the equation IN1+IN2=Q.	All except S, SA, SB, SC	No
Q	The result of IN1+IN2. If an ADD of signed operands results in overflow, Q is set to the largest possible value and there is no power flow. If an ADD_UINT operation results in overflow, Q wraps around.	All except S, SA, SB, SC and constant.	No

Examples for ADD

The first example is a failed attempt to create a counter circuit that would count the number of times switch %I0001 closes. The running total is stored in register %R0002. The intent of this design is that when %I0001 closes, the ADD instruction should add one to the value in %R0002 and place the new value right back into %R0002. The problem with this design is that the ADD instruction executes once every PLC scan while %I0001 is closed. For example, if %I0001 stays closed for five scans, the output increments five times, even though %I0001 only closed once during that period.



To correct the above problem, the enable input to the ADD instruction should come from a transition (“one-shot”) coil, as shown below. In the improved circuit, the %I0001 input switch controls a transition coil, %M0001, whose contact turns on the enable input of the ADD function for only one scan each time contact %I0001 closes. In order for the %M0001 contact to close again, contact %I0001 has to open and close again.

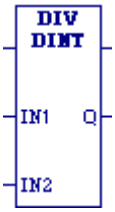


Note: If IN1 and/or IN2 is NaN (Not a Number), ADD_REAL passes no power flow.

Divide

When the DIV function receives power flow, it divides the operand IN1 by the operand IN2 of the same data type as IN1 and stores the quotient in the output variable assigned to Q, also of the same data type as IN1 and IN2.

The power flow output is energized when DIV is performed, unless an invalid operation or overflow occurs. (For more information, see “Overflow” on page 8-115.)



Mnemonics:
DIV_DINT
DIV_INT
DIV_MIXED
DIV_REAL
DIV_LREAL
DIV_UINT

Notes:

- DIV rounds down; it does not round to the closest integer. For example, 24 DIV 5 = 4.
- DIV_MIXED uses mixed data types.
- Be careful to avoid overflows.

The following REAL and LREAL operations are invalid for DIV:

- Any number divided by 0. This operation yields a result of 65535.
- ∞ divided by ∞
- I1 and/or I2 is NaN (Not a Number)

Mnemonic	*Operation*	*Displays as*
DIV_UINT	Q(16 bit) = IN1(16 bit) / IN2(16 bit)	base 10 number, unsigned, up to 5 digits long
DIV_INT	Q(16 bit) = IN1(16 bit) / IN2(16 bit)	base 10 number with sign, up to 5 digits long
DIV_DINT	Q(32 bit) = IN1(32 bit) / IN2(32 bit)	base 10 number with sign, up to 10 digits long
DIV_MIXED	Q(16 bit) = IN1(32 bit) / IN2(16 bit)	base 10 number with sign, up to 5 digits long
DIV_REAL	Q(32 bit) = IN1(32 bit) / IN2(32 bit)	base 10 number, sign and decimals, up to 8 digits long (excluding the decimals)
DIV_LREAL	Q(64 bit) = IN1(64 bit) / IN2(64 bit)	base 10 number, sign and decimals, up to 17 digits long (excluding the decimals)

Operands for the DIV Function

Parameter	*Description*	*Allowed Operands*	*Optional*
IN1	The value to be divided; the value to the left of “DIV” in the equation IN1 DIV IN2=Q.	All except S, SA, SB, SC	No
IN2	The value to divide IN1 with; the value to the right of “DIV” in the equation IN1 DIV IN2=Q.	All except S, SA, SB, SC	No
Q	The quotient of IN1/IN2. If a DIV operation on signed operands results in overflow, Q is set to the largest possible value and there is no power flow. If a DIV_UINT operation results in overflow, Q wraps around.	All except S, SA, SB, SC and constant	No

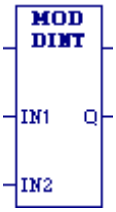
DIV_MIXED Operands

Parameter	Description	Allowed Operands	Optional
IN1	The value to be divided; the value to the left of "DIV" in the equation IN1 DIV IN2=Q.	All except S, SA, SB, SC	No
IN2	The value to divide IN1 with; the value to the right of "DIV" in the equation IN1 DIV IN2=Q.	All except S, SA, SB, SC	No
Q	The quotient of IN1/IN2. If an overflow occurs, the result is the largest value with the proper sign and no power flow.	All except S, SA, SB, SC and constant	No

DIV_MIXED Example

DIV_DINT can be used in conjunction with a MUL_DINT function to scale a ±10 volt input to ±25,000 engineering units. See "Example – Scaling an Analog Input" on page 8-121.

Modulus



Mnemonics:
DIV_DINT
MOD_INT
MOD_UINT

When the Modulo Division (MOD) function receives power flow, it divides input IN1 by input IN2 and outputs the remainder of the division to Q.

All three operands must be of the same data type. The sign of the result is always the same as the sign of input parameter IN1. Output Q is calculated using the formula:

Q = IN1-((IN1 DIV IN2) * IN2)

where DIV produces an integer number.

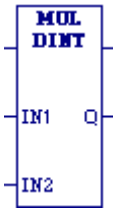
The power flow output is always ON when the function receives power flow, unless there is an attempt to divide by zero. In that case, the power flow output is set to OFF.

Operands for Modulus Function

Parameter	Description	Allowed Operands	Optional
IN1	The value to be divided to obtain the remainder; the value to the left of "MOD" in the equation IN1 MOD IN2=Q.	All except S, SA, SB, SC	No
IN2	The value to divide IN1 with; the value to the right of "MOD" in the equation IN1 MOD IN2=Q.	All except S, SA, SB, SC	No
Q	The remainder of IN1/IN2.	All except S, SA, SB, SC and constant	No

Multiply

When the MUL function receives power flow, it multiplies the two operands IN1 and IN2 of the same data type and stores the result in the output variable assigned to Q, also of the same data type.



Mnemonics:
MUL_DINT
MUL_INT
MUL_MIXED
MUL_REAL
MUL_LREAL
MUL_UINT

The power flow output is energized when the function is performed, unless an invalid operation or overflow occurs. (For more information, see “Overflow” on page 8-115.)

Note: MUL_MIXED uses mixed data types. Be careful to avoid overflows.

The following REAL and LREAL operations are invalid for MUL:

- 0 x ∞
- I1 and/or I2 is NaN (Not a Number).

Mnemonic	*Operation*	*Displays as*
MUL_INT	Q(16 bit) = IN1(16 bit) * IN2(16 bit)	base 10 number with sign, up to 5 digits long
MUL_DINT	Q(32 bit) = IN1(32 bit) * IN2(32 bit)	base 10 number with sign, up to 10 digits long
MUL_REAL	Q(32 bit) = IN1(32 bit) * IN2(32 bit)	base 10 number, sign and decimals, up to 8 digits long (excluding the decimals)
MUL_LREAL	Q(64 bit) = IN1(64 bit) * IN2(64 bit)	base 10 number, sign and decimals, up to 17 digits long (excluding the decimals)
MUL_UINT	Q(16 bit) = IN1(16 bit) * IN2(16 bit)	base 10 number, unsigned, up to 5 digits long
MUL_MIXED	Q(32 bit) = IN1(16 bit) * IN2(16 bit)	base 10 number with sign, up to 10 digits long

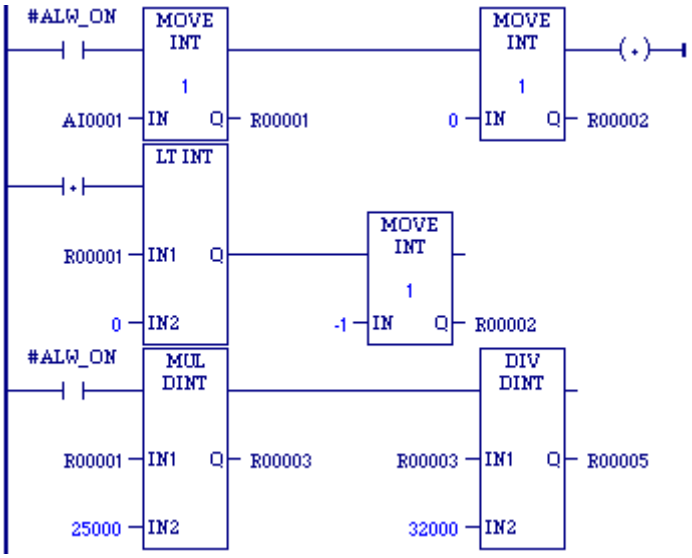
Operands for Multiply

Parameter	*Description*	*Allowed Operands*	*Optional*
IN1	The first value to multiply; the value to the left of the multiply sign (*) in the equation IN1 * IN2=Q.	All except S, SA, SB, SC	No
IN2	The second value to multiply; the value to the right of the multiply sign (*) in the equation IN1 * IN2=Q.	All except S, SA, SB, SC	No
Q	The result of IN1*IN2. If a MUL operation on signed operands results in overflow, Q is set to the largest possible value and there is no power flow. If a MUL_UINT operation results in overflow, Q wraps around.	All except S, SA, SB, SC and constant	No

Example – Scaling Analog Input Values

A common application is to scale analog input values with a MUL operation followed by a DIV and possibly an ADD operation. A 0 to ±10 volt analog input will place values of 0 to ±32,000 in its corresponding %AI input register. Multiplying this input register using an MUL_INT function will result in an overflow since an INT type instruction has an input and output range of 32,767 to –32,768. Using the %AI value as in input to a MUL_DINT also does not work as the 32-bit IN1 will combine 2 analog inputs at the same time. To solve this problem, you can move the analog input to the low word of a double register, then test the sign and set the second register to 0 if the sign tests positive or –1 if negative. Then use the double register just created with a MUL_DINT which gives a 32-bit result, and which can be used with a following DIV_DINT function.

For example, the following logic could be used to scale a ±10 volt input %AI1 to ±25000 engineering units in %R5.

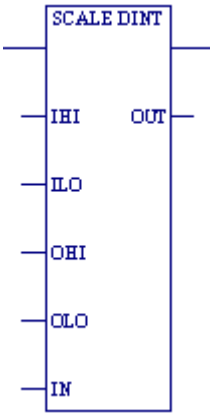


An alternate, but less accurate, way of programming this circuit using INT values involves placing the DIV_DINT instruction first, followed by the MUL_DINT instruction. The value of IN2 for the DIV instruction would be 32, and the value of IN2 for the MUL would be 25. This maintains the scaling proportion of the above circuit and keeps the values within the working range of the INT type instructions. However, the DIV instruction inherently discards any remainder value, so when the DIV output is multiplied by the MUL instruction, the error introduced by a discarded remainder is multiplied. The percent of error is non-linear over the full range of input values and is greater at lower input values.

By contrast, in the example above, the results are more accurate because the DIV operation is performed last, so the discarded remainder is not multiplied. If even greater precision is required, substitute REAL type math instructions in this example so that the remainder is not discarded.

Scale

When the SCALE function receives power flow, it scales the input operand IN and places the result in the output variable assigned to output operand OUT. The power flow output is energized when SCALE is performed without overflow.



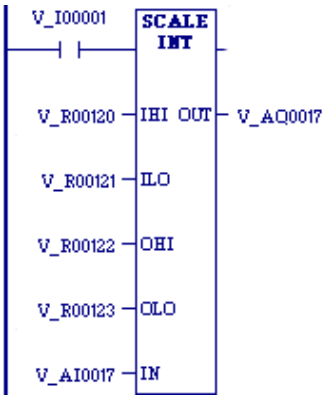
Mnemonics:
SCALE_DINT
SCALE_INT
SCALE_DINT
SCALE_UINT

Operands

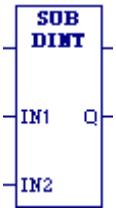
Parameter	*Description*	*Allowed Operands*	*Optional*
IHI	(Inputs High) Maximum input value (module-related). The upper limit of the unscaled data. IHI is used with ILO, OHI and OLO to calculate the scaling factor applied to the input value IN.	All except S, SA, SB, SC	No
ILO	(Inputs Low) Minimum input value (module-related). The lower limit of the unscaled data. Must be the same data type as IHI.	All except S, SA, SB, SC	No
OHI	(Outputs High) Maximum output value. The upper limit of the scaled data. Must be the same data type as IHI. When the IN input is at the IHI value, the OUT value is the same as the OHI value.	All except S, SA, SB, SC	No
OLO	(Outputs Low) Minimum output value. The lower limit of the scaled data. Must be the same data type as IHI. When the IN input is at the ILO value, the OUT value is the same as the OLO value.	All except S, SA, SB, SC	No
IN	(INput value) The value to be scaled. Must be the same data type as IHI	All except S, SA, SB, SC	No
OUT	(OUTput value) The scaled equivalent of the input value. Must be the same data type as IHI.	All except S, SA, SB, SC	No

Example

In the example, the registers %R0120 through %R0123 are used to store the high and low scaling values. The input value to be scaled is analog input %AI0017. The scaled output data is used to control analog output %AQ0017. The scaling is performed whenever %I0001 is ON.



Subtract



Mnemonics:
SUB_DINT
SUB_INT
SUB_REAL
SUB_LREAL
SUB_UINT

When the SUB function receives power flow, it subtracts the operand IN2 from the operand IN1 of the same data type as IN2 and stores the result in the output variable assigned to Q, also of the same data type.

The power flow output is energized when SUB is performed, unless an invalid operation or overflow occurs. (For more information, see “Overflow” on page 8-115.)

If a SUB_UINT operation results in a negative number, Q wraps around, yielding a result that is the highest possible value (65535) minus the absolute value of the difference -1.

The following REAL and LREAL operations are invalid for SUB:

- (± ∞) – (± ∞)
- I1 and/or I2 is NaN (Not a Number)

Mnemonic	*Operation*	*Displays as*
SUB_INT	Q(16 bit) = IN1(16 bit) – IN2(16 bit)	base 10 number with sign, up to 5 digits long
SUB_DINT	Q(32 bit) = IN1(32 bit) – IN2(32 bit)	base 10 number with sign, up to 10 digits long
SUB_REAL	Q(32 bit) = IN1(32 bit) – IN2(32 bit)	base 10 number, sign and decimals, up to 8 digits long (excluding the decimals)
SUB_LREAL	Q(64 bit) = IN1(64 bit) – IN2(64 bit)	base 10 number, sign and decimals, up to 17 digits long (excluding the decimals)
SUB_UINT	Q(16 bit) = IN1(16 bit) – IN2(16 bit)	base 10 number, unsigned, up to 5 digits long

Operands for Subtract

Parameter	*Description*	*Allowed Operands*	*Optional*
IN1	The value to subtract from; the value to the left of the minus sign (-) in the equation IN1-IN2=Q.	All except S, SA, SB, SC	No
IN2	The value to subtract from IN1; the value to the right of the minus sign (-) in the equation IN1-IN2=Q.	All except S, SA, SB, SC	No
Q	The result of IN1-IN2. If a SUB operation on signed operands results in underflow, Q is set to the smallest possible value and there is no power flow. If a SUB_UINT operation results in overflow, Q wraps around. For example, The SUB_UINT operation 600 – 601 = –1 sets Q to 65535 The SUB_UINT operation 600 – 602 = –2 sets Q to 65534	All except S, SA, SB, SC and constant	No

Program Flow Functions

The program flow functions limit program execution or change the way the CPU executes the application program.

Function	Mnemonic	Description
Argument Present	ARG_PRES	Determines whether an input or output parameter value was present when the function block instance of the parameter was invoked. For example, a parameter can be optional (pass by value).
Call	CALL	Causes program execution to go to a specified block.
Comment	COMMENT	Places a text explanation in the program.
End Master Control Relay	ENDMCRN	Nested End Master Control Relay. Indicates that the subsequent logic is to be executed with normal power flow.
End of Logic	END	Provides an unconditional end of logic. The program executes from the first rung to the last rung or the END instruction, whichever is encountered first.
Jump	JUMPN	Nested jump. Causes program execution to jump to a specified location indicated by a LABELN. JUMPN/LABELN pairs can be nested within one another. Multiple JUMPNs can share the same LABELN.
Label	LABELN	Nested label. Specifies the target location of a JUMPN instruction.
Master Control Relay	MCRN	Nested Master Control Relay. Causes all rungs between the MCR and its subsequent ENDMCRN to be executed without power flow. Up to MCRN/ENDMCRN pairs can be nested within one another. All the MCRNs share the same ENDMCRN.
Wires	H_WIRE	Horizontally connects elements of a line of LD logic, to complete the power flow.
	V_WIRE	Vertically connects elements of a line of LD logic, to complete the power flow.

Argument Present

The ARG_PRES function determines whether an input parameter value was present when the function block instance of the parameter was invoked. This may be necessary if the parameter is optional.



This function must be called from a function block instance or a parameterized block.

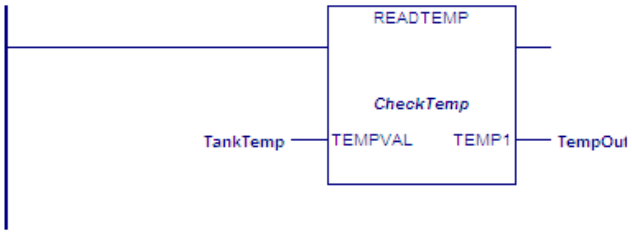
The standard output parameter ENO is false only when EN is false.

Operands for ARG_PRES

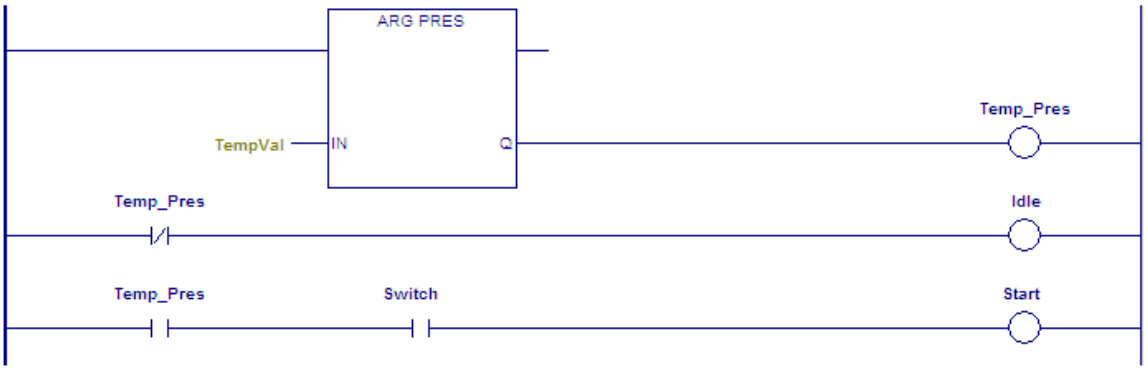
Parameter	Description	Allowed Operands	Optional
IN	Parameter name. Must be a parameter of the function block that contains the ARG_PRES instruction. Cannot be an array element or structure element. An alias to a parameter should resolve only to the parameter name.	All except flow and constants.	No
Q	True if the parameter is present, otherwise false.	Must be flow in LD. In other languages all types allowed except S, SA, SB, SC and constants.	No

Example for ARG_PRES

The following sample rung calls the user defined function block, ReadTemp, which has two parameters, TempVal and Temp1.

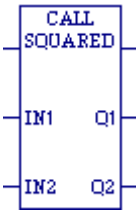


The function block ReadTemp contains the following logic, which uses an ARG_PRES function to determine whether a value for TempVal is present. If TempVal does not have a value, Temp_Pres is OFF and Idle is ON. If a value exists for TempVal, the ARG_PRES function sets Temp_Pres ON. When Temp_Pres and Switch are both ON, Start is set ON.



Call





Non-parameterized **Parameterized.** May call a parameterized external block or a parameterized block. May have up to 7 input and 8 output parameters.

When the CALL function receives power flow, it causes the logic execution to goimmediately to the designated program block, external C block (parameterized or not), or parameterized block and execute it. After the block's execution is complete, control returns to the point in the logic immediately following the CALL instruction.

Notes:

- A CALL function can be used in any program block, including the _MAIN block, or a parameterized block. It cannot be used in an external block.
- You cannot call a _MAIN block.
- The called block must exist in the target before making the call.
- There is no limit to the number of calls that can be made from or to a given block.
- You can set up recursive subroutines by having a block call itself. When stack size is configured to be the default (64K), the PLC guarantees a minimum of eight nested calls before an **"**Application Stack Overflow**"** fault is logged.
- Each block has a predefined parameter, Y0, which the CPU sets to 1 upon each invocation of the block. Y0 can be controlled by logic within the block and provides the output status of the block. When the Y0 parameter of a Program Block, parameterized block, or external C block returns ON, the CALL passes power to the right; when it returns OFF, the CALL does not pass power to the right.

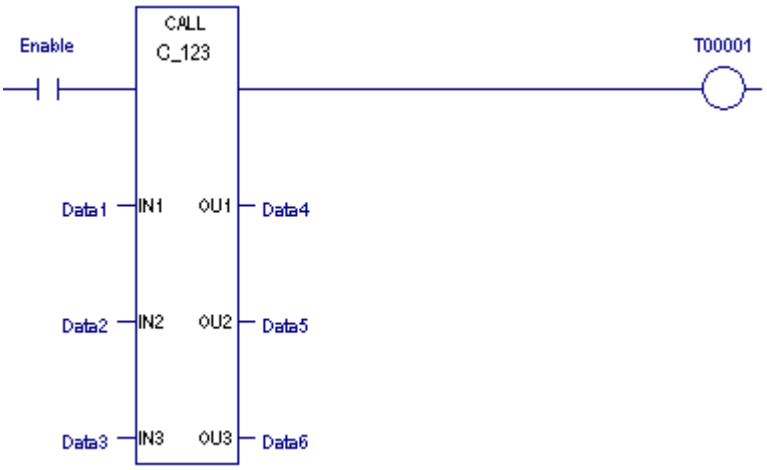
Operands for Call

Parameter	Description
Block Name (????)	Block name; the name of the block to transfer to. You cannot CALL the _MAIN block. A program block or a parameterized block can call itself.
(Parameterized calls only) Input parameters (0 – 7) Output parameters (1 – 8)	**Notes for External (C) blocks:** ■ You must define the TYPE, LENGTH, and NAME for each external C block parameter. ■ The valid data type, value range, and memory area for each parameter are stated in the external block's written documentation. ■ Data flow is permitted for any parameter. ■ For additional information, see the section on External Blocks in chapter 6. **Notes for Parameterized Blocks:** ■ You must define the TYPE, LENGTH, and NAME for each parameter. Valid operands on the CALL instruction include variables, flow, and indirect references. Input operands can also be constants. ■ If a formal parameter is an array of BOOL type and has a length evenly divisible by 16, then a variable or array residing in word-oriented memory can be passed on to the parameterized block as an operand. For example, if a parameterized block has a formal parameter Y1 of data type BIT and length 48, you can pass a WORD array of length 3 to Y1. ■ The BOOL parameter Y0 is automatically defined for all parameterized blocks and can be used in the parameterized block's logic. When the parameterized block stops executing and Y0 is ON, the CALL passes power flow to the right. If Y0 is OFF, the CALL passes no power flow. ■ A parameterized block is not required to have the same number of inputs and outputs. ■ For additional information, see “Using Parameters With a Parameterized Block” in chapter 6.

Examples for Call

Example 1

In the following example, if Enable is set, the C block named C_123 is executed. C_123 operates on the input data located at reference addresses Data1, Data2, and Data 3, and produces values located at reference addresses Data4, Data5, and Data6. Logic within C_123 controls the power flow output.



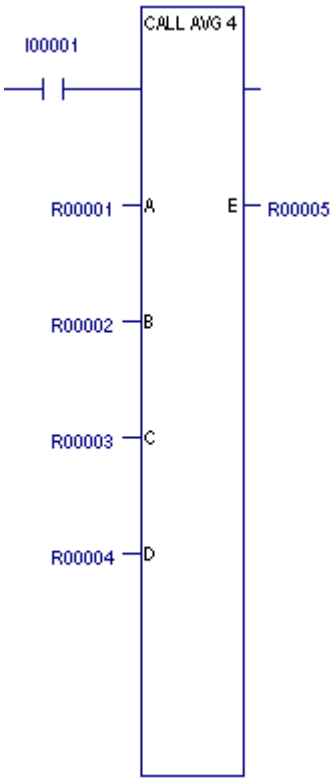
Example 2

Parameterized blocks are useful for building libraries of user-defined functions. For example, if you have an equation such as:

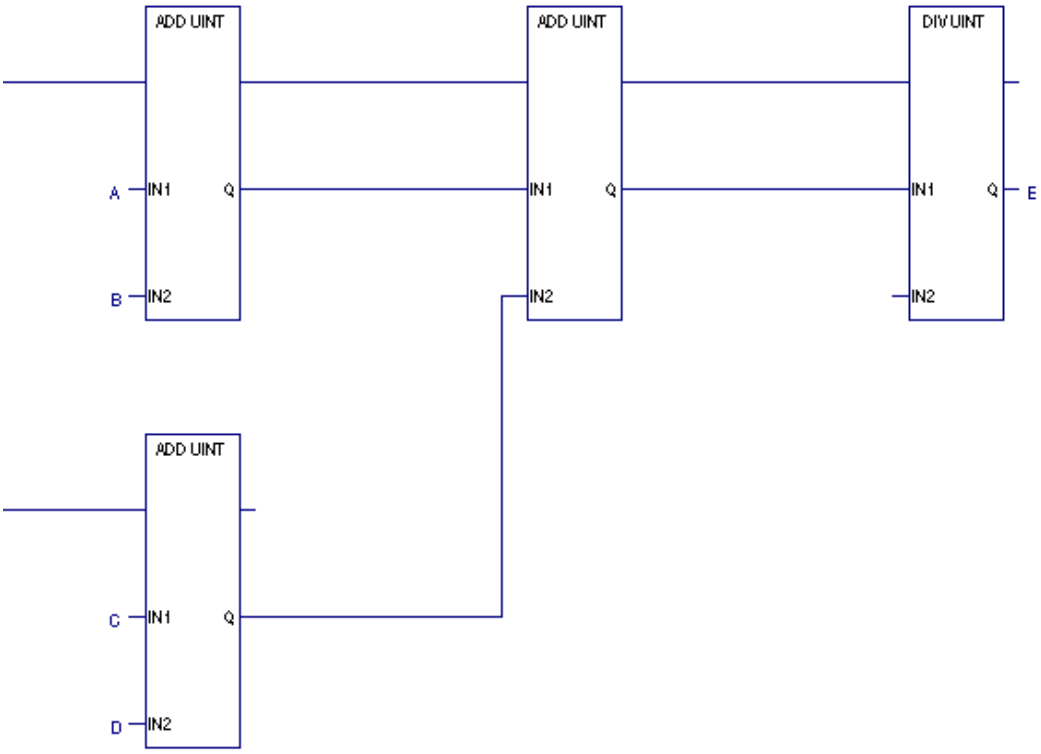
E=(A+B+C+D)/4, a parameterized block named AVG_4 could be called as shown in the example to the right.

In this example, the average of the values in R00001, R00002, R00003, and R00004 would be placed in R00005.

The logic within the parameterized block would be defined as shown below.



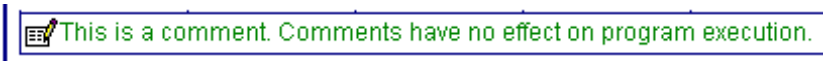
Logic for AVG_4 Parameterized Block



Comment



The Comment function is used to enter a text explanation in the program. When you insert a Comment instruction into the LD logic, it displays ????. After you key in a comment, the first few words are displayed.



You can set the Comment mode option to Brief or Full.

Notes:

- In Microsoft ® Windows ® 9x, the maximum length is 32,766 characters (0x7FFE). In Microsoft ® Windows 2000 ® or NT ®, the maximum length of a comment is 2GB - 1 byte, for a total of 2,147,483,646 characters (0x7FFFFFFE).
- Since comments are not downloaded to the PLC, you can edit comments, whether offline or online, without losing equality.

Jump

Mnemonic	Description	Always associated with...
JUMPN	Nested form of Jump instruction.	a LABELN instruction



A JUMPN instruction causes a portion of the program logic to be bypassed. Program execution continues at the LABELN specified in the same block. Power flow jumps directly from the JUMPN to the rung with the named LABELN.

When the Jump is active, any functions between the jump and the label are not executed. All coils between JUMPN and its associated LABELN are left at their previous states. This includes coils associated with timers, counters, latches, and relays.

Any JUMPN can be either a forward or a backward jump, i.e., its LABELN can be either in a further or previous rung. The LABELN must be in the same block.

Note: To avoid creating an endless loop with forward and backward JUMPN instructions, a backward JUMPN should contain a way to make it conditional.

A JUMPN and its associated LABELN can be placed anywhere in a program, as long as the JUMPN / LABELN range:

- does not overlap the range of a MCRN / ENDMCRN pair.
- does not overlap the range of a FOR_LOOP / END_FOR pair.

Nothing can be connected to the right side of a JUMPN instruction.

Operands

Parameter	Description	Optional
Label (????)	Label name; the name assigned to the destination LABEL(N).	No

Master Control Relay/End Master Control Relay



Mnemonics	Description	Always associated with...
MCRN	Nested form of the Master Control Relay	an ENDMCRN instruction
ENDMCRN	Nested End Master Control Relay	an MCRN instruction

MCRN

An MCRN instruction marks the beginning of a section of logic that will be executed with no power flow. The end of an MCRN section must be marked with an ENDMCRN having the same name as the MCRN. ENDMCRNs must follow their corresponding MCRNs in the logic.

All rungs between an active MCRN and its corresponding ENDMCRN are executed with negative power flow from the power rail. The ENDMCRN function associated with the MCRN causes normal program execution to resume, with positive power flow coming from the power rail.

With a Master Control Relay, functions within the scope of the Master Control Relay are executed *without power flow*, and *coils are turned off*.

Block calls within the scope of an active Master Control Relay will not execute. However, any timers in the block will continue to accumulate time.

A rung may not contain anything after an MCRN.

Unlike JUMP instructions, MCRNs can only move forward. An ENDMCRN instruction must appear after its corresponding MCRN instruction in a program.

The following controls are imposed by an MCRN:

- Timers do not increment or decrement. TMR types are reset. For an ONDTR function, the accumulator holds its value.
- Normal outputs are off; negated outputs are on.

Note: When an MCRN is energized, the logic it controls is scanned and contact status is displayed, but no outputs are energized. If you are not aware that an MCRN is controlling the logic being observed, this might appear to be a faulty condition.

An MCRN and its associated ENDMCRN can be placed anywhere in a program, as long as the MCRN / ENDMCRN range:

- Is completely nested within another MCRN / ENDMCRN range, up to a maximum 255 levels of nesting, or is completely outside of the range of another MCRN / ENDMCRN range.
- Is completely nested within a FOR_LOOP / END_FOR range or is completely outside of the range of a FOR_LOOP / END_FOR.

EndMCRN

The End Master Control Relay instruction marks the end of a section of logic begun with a Master Control Relay instruction. When the MCRN associated with the ENDMCRN is active, the ENDMCRN causes program execution to resume with normal power flow. When the MCRN associated with the ENDMCRN is not active, the ENDMCRN has no effect.

ENDMCRN must be tied to the power rail; there can be no logic before it in the rung; execution cannot be conditional.

ENDMCRN has a name that identifies it and associates it with the corresponding MCRN(s). The ENDMCRN function has no outputs; there can be nothing after an ENDMCR instruction in a rung.

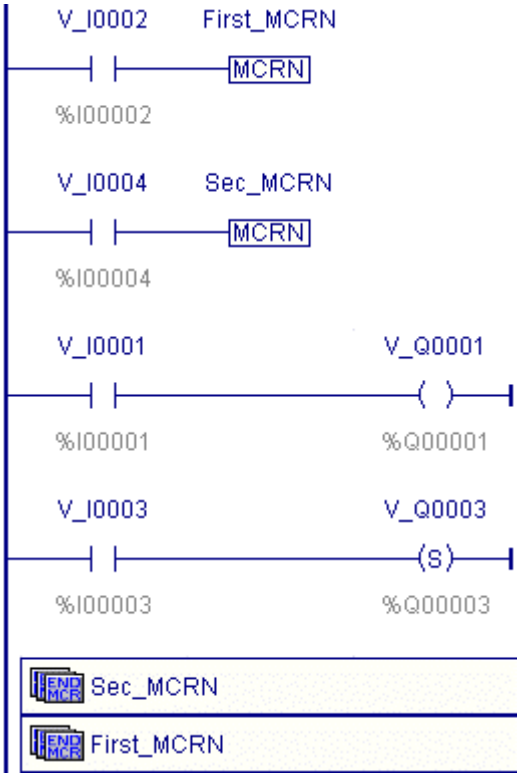
Operands for MCRN/ENDMCRN

The Master Control Relay function has a single operand, a name that identifies the MCRN. This name is used again with an ENDMCRN instruction. The MCRN has no output.

Parameter	*Description*	*Optional*
Name (????)	The name associated with the MCRN that starts the section of logic.	No

Example of MCRN/ENDMCRN

The following example shows an MCRN named **"**Sec_MCRN**"** nested inside the MCRN named **"**First_MCRN.**"** Whenever the V_I0002 contact allows power flow into the MCRN function, program execution will continue without power flow to the coils until the associated ENDMCRN is reached. If the V_I0001 and V_I0003 contacts are ON, the V_Q0001 coil is turned OFF and the SET coil V_Q0003 maintains its current state.



Wires

Horizontal and vertical wires (H_WIRE and V_WIRE) are used to connect elements of a line of LD logic between functions. Their purpose is to complete the flow of logic (“power”) from left to right in a line of logic.



A horizontal wire transmits the BOOLEAN ON/OFF state of the element on its immediate left to the element on its immediate right.

A vertical wire may intersect with one or more horizontal wires on each side. The state of the vertical wire is the inclusive OR of the ON states of the horizontal wires on its left side. The state of the vertical wire is copied to all of the attached horizontal wires on its right side.

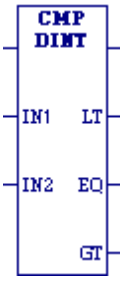
Note: Wires can be used for data flow, but you cannot route data flow leftwards. Nor can two separate data flow lines come into the left side of the same vertical wire.

Relational Functions

Relational functions compare two values of the same data type or determine whether a number lies within a specified range. The original values are unaffected.

Function	*Mnemonic*	*Description*
Compare	CMP_DINT CMP_INT CMP_REAL CMP_LREAL CMP_UINT	Compares two numbers, IN1 and IN2, of the data type specified by the mnemonic. ■ If IN1 < IN2, the LT output is turned ON. ■ If IN1 = IN2, the EQ output is turned ON. ■ If IN1 > IN2, the GT output is turned ON.
Equal	EQ_DINT EQ_INT EQ_REAL EQ_LREAL EQ_UINT	Tests two numbers for equality
Greater or Equal	GE_DINT GE_INT GE_REAL GE_LREAL GE_UINT	Tests whether one number is greater than or equal to another
Greater Than	GT_DINT GT_INT GT_REAL GT_LREAL GT_UINT	Tests whether one number is greater than another
Less or Equal	LE_DINT LE_INT LE_REAL LE_LREAL LE_UINT	Tests whether one number is less than or equal to another
Less Than	LT_DINT LT_INT LT_REAL LT_LREAL LT_UINT	Tests whether one number is less than another
Not Equal	NE_DINT NE_INT NE_REAL NE_LREAL NE_UINT	Tests two numbers for non-equality
Range	RANGE_DINT RANGE_DWORD RANGE_INT RANGE_UINT RANGE_WORD	Tests whether one number is within the range defined by two other supplied numbers

Compare



Mnemonics:

CMP_DINT
CMP_INT
CMP_REAL
CMP_LREAL
CMP_UINT

When the Compare (CMP) function receives power flow, it compares the value IN1 to the value IN2.

- If IN1 < IN2, CMP energizes the LT (Less Than) output.
- If IN1 = IN2, CMP energizes the EQ (Equal) output.
- If IN1 > IN2, CMP energizes the GT (Greater Than) output.

IN1 and IN2 must be the same data type. CMP compares data of the following types: DINT, INT, REAL, LREAL, and UINT.

Tip: To compare values of different data types, first use conversion functions to make the types the same.

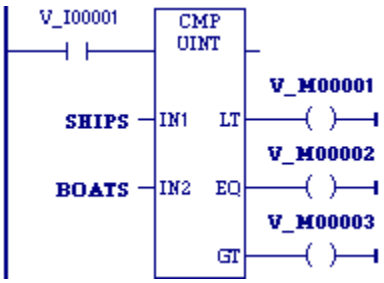
When it receives power flow, CMP always passes power flow to the right, unless IN1 and/or IN2 is NaN (Not a Number).

Operands

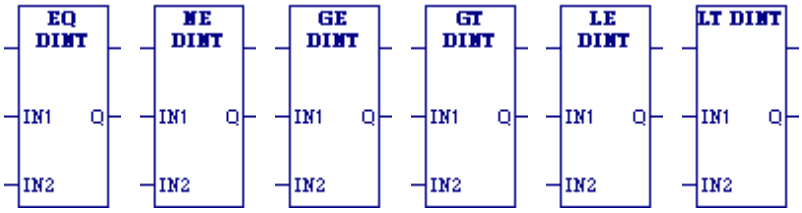
Parameter	*Description*	*Allowed Operands*	*Optional*
IN1	The first value to compare.	All except S, SA, SB, SC	No
IN2	The second value to compare.	All except S, SA, SB, SC	No
LT	Output LT is energized when I1 < I2.	Power flow	No
EQ	Output EQ is energized when I1 = I2.	Power flow	No
GT	Output GT is energized when I1 > I2.	Power flow	No

Example

When %I00001 is ON, the integer variable SHIPS is compared with the variable BOATS. Internal coils %M0001, %M0002, and %M0003 are set to the results of the compare.



Equal, Not Equal, Greater or Equal, Greater Than, Less or Equal, Less Than



Other data types:
_INT
_REAL
_LREAL
_UINT

When the relational function receives power flow, it compares input IN1 to input IN2. These operands must be the same data type. If inputs IN1 and IN2 are equal, the function passes power to the right, unless IN1 and/or IN2 is NaN (Not a Number). The following relational functions can be used to compare two numbers:

Function	*Definition*	*Relational Statement*
EQ	Equal	IN1=IN2
NE	Not Equal	IN1≠IN2
GE	Greater Than or Equal	IN1≥IN2
GT	Greater Than	IN1>IN2
LE	Less Than or Equal	IN1≤IN2
LT	Less Than	IN1<IN2

Note: If an overflow occurs with a _UINT operation, the result wraps around – see “Overflow” on page 8-115.

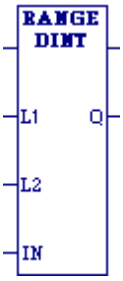
If the _DINT or _INT operations are fed the largest possible value with any sign, they cannot determine if it is an overflow value. The power flow output of the previous operation would need to be checked. If an overflow occurred on a previous DINT, or INT operation, the result was the largest possible value with the proper sign and no power flow.

Tip: To compare values of different data types, first use conversion functions to make the types the same. The relational functions require data to be one of the following types: DINT, INT, REAL, LREAL, or UINT.

Operands

Parameter	*Description*	*Allowed Operands*	*Optional*
IN1	The first value to be compared; the value on the left side of the relational statement.	All except S, SA, SB, SC	No
IN2	The second value to be compared; the value on the right side of the relational statement. IN2 must be the same data type as IN1.	All except S, SA, SB, SC	No
Q	The power flow. If the relational statement is true, Q is energized, unless IN1 or IN2 is NaN.	Power flow	No

Range



Mnemonics:
RANGE_DINT
RANGE_DWORD
RANGE_INT
RANGE_UINT
RANGE_WORD

When the Range function is enabled, it compares the value of input IN against the range delimited by operands L1 and L2. Either L1 or L2 can be the high or low limit. When L1 ≤ IN ≤ L2 or L2 ≤ IN ≤ L1, output parameter Q is set ON (1). Otherwise, Q is set OFF (0).

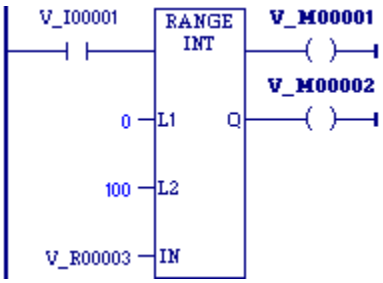
If the operation is successful, it passes power flow to the right.

Operands

Parameter	Description	Allowed Operands	Optional
IN	The value to compare against the range delimited by L1 and L2. Must be the same data type as L1 and L2.	All except S, SA, SB, SC	No
L1	The start point of the range. May be the upper limit or the lower limit. Must be the same data type as IN and L2.	All except S, SA, SB, SC	No
L2	The end point of the range. May be the lower or upper limit. Must be the same data type as IN and L1.	All except S, SA, SB, SC	No
Q	If L1 ≤ IN ≤ L2 or L2 ≤ IN ≤ L1, Q is energized; otherwise, Q is off.	Power flow	No

Example

When RANGE_INT receives power flow from the normally open contact %I0001, it determines whether the value in %R00003 is within the range 0 to 100 inclusively. Output coil %M00002 is ON only if 0 ≤ %AI0050 ≤ 100.



Timers

This section describes the PACSystems timed contacts and timer function blocks that are implemented in the LD language.

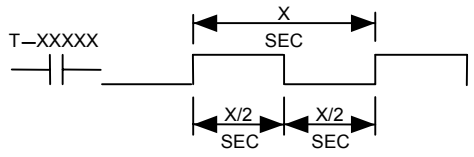
Timed Contacts

The PACSystems has four timed contacts that can be used to provide regular pulses of power flow to other program functions. Timed contacts cycle on and off, in square-wave form, every 0.01 second, 0.1 second, 1.0 second, and 1 minute. Timed contacts can be read by an external communications device to monitor the state of the CPU and the communications link. Timed contacts are also often used to blink pilot lights and LEDs.

The timed contacts are referenced as T_10MS (0.01 second), T_100MS (0.1 second), T_SEC (1.0 second), and T_MIN (1 minute). These contacts represent specific locations in %S memory:

#T_10MS	0.01 second timed contact	%S0003
#T_100MS	0.1 second timed contact	%S0004
#T_SEC	1.0 second timed contact	%S0005
#T_MIN	1.0 minute timed contact	%S0006

These contacts provide a pulse having an equal on and off time duration. The following timing diagram illustrates the on/off time duration of these contacts.



Caution

Do not use timed contacts for applications requiring accurate measurement of elapsed time. Timers, time-based subroutines, and PID blocks are preferred for these types of applications.

The CPU updates the timed contact references based on a free-running timer that has no relationship to the start of the CPU sweep. If the sweep time remains in phase with the timed contact clock, the contact will always appear to be in the same state. For example, if the CPU is in constant sweep mode with a sweep time setting of 100ms, the T_10MS and T_100MS bits will never toggle.

Note: For a summary of differences in the operation of timed contacts in PACSystems CPUs compared to Series 90-70 and Series 90-30, see “LD Function Differences” in appendix C.

Timer Function Blocks

Function	Function Block Type	Mnemonic	Description
Off Delay Timer	Built-in (instance data is WORD array) See page 8-140.	OFDT_HUNDS OFDT_SEC OFDT_TENTHS OFDT_THOUS	The timer's Current Value (CV) resets to zero when power flow input is on. CV increments while power flow is off. When CV=PV (Preset Value), power flow is no longer passed to the right until power flow input is on again.
On Delay Stopwatch Timer		ONDTR_HUNDS ONDTR_SEC ONDTR_TENTHS ONDTR_THOUS	Retentive on delay timer. Increments while it receives power flow and holds its value when power flow stops.
On Delay Timer		TMR_HUNDS TMR_SEC TMR_TENTHS TMR_THOUS	Simple on delay timer. Increments while it receives power flow and resets to zero when power flow stops.
Timer Off Delay	Standard (instance data is a structure variable) See page 8-151.	TOF	When the input IN transitions from ON to OFF, the timer starts timing until a specified period of time has elapsed, then sets the output Q to OFF.
Timer On Delay		TON	When the input IN transitions from OFF to ON, the timer starts timing until a specified period of time has elapsed, then sets the output Q to ON.
Timer Pulse		TP	When the input IN transitions from OFF to ON, the timer sets the output Q to ON for a specified time interval.

Built-In Timer Function Blocks

Note: Special care must taken when programming timers in parameterized blocks. For details, see “Using OFDT, ONDTR and TMR in Parameterized Blocks” on page 8-141.

Data Required for Built-in Timer Function Blocks

The data associated with these functions is retentive through power cycles. Each timer uses a one-dimensional, three-word array of %R, %W, %P, %L, or symbolic memory to store the following information:

Current value (CV)	Word 1
Preset value (PV)	Word 2
Control word	Word 3

When you program a timer, you must enter a beginning address for the three-word array (three-word block of registers).

Warning

Do not use two consecutive words (registers) as the starting addresses of two timers. Logic Developer - PLC does not check or warn you if register blocks overlap. Timers will not work if you place the current value of a second timer on top of the preset value for the previous timer.

Word 1: Current value (CV)

Warning

The first word (CV) can be read but should not be written to, or the function may not work properly.

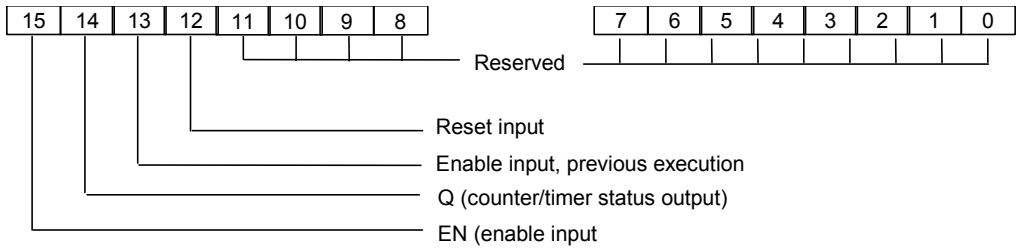
Word 2: Preset value (PV)

When the Preset Value (PV) operand is a variable, it is normally set to a different location than word 2 in the timer's or counter's three-word array.

- If you use a different address and you change word 2 directly, your change will have no effect, as PV will overwrite word 2.
- If you use the same address for the PV operand and word 2, you can change the Preset Value in word 2 while the timer or counter is running and the change will be effective.

Word 3: Control word

The control word stores the state of the Boolean inputs and outputs of its associated timer or counter, as shown in the following diagram:



Warning

The third word (Control) can be read but should not be written to; otherwise, the function will not work.

Note: Bits 0 through 13 are used for timer accuracy.

Using OFDT, ONDTR and TMR in Parameterized Blocks

Special care must taken when programming timers in PACSystems parameterized blocks. Timers in parameterized blocks can be programmed to track true real-time as long as the guidelines and rules below are followed. If the guidelines and rules described here are not followed, the operation of the timer functions in parameterized blocks is undefined.

Note: These rules are not enforced by the programming software. It is your responsibility to ensure these rules are followed.

The best use of a timer function is to invoke it with a particular reference address exactly one time each scan. With parameterized blocks, it is important to use the appropriate reference memory with the timer function and to call the parameterized block an appropriate number of times.

Finding the Source Block

The source block is either the _MAIN block or the lowest logic block of type Block that appears above the parameterized block in the call tree. To determine the source block for a given parameterized block, determine which block invoked that parameterized block. If the calling block is _MAIN or of type Block, it is the source block. If the calling block is any other type (parameterized block or function block), apply the same test to the block that invoked this block. Continue back up the call tree until the _MAIN block or a block of type Block is found. This is the source block for the parameterized block.

Programming OFDT, ONDTR and TMR in Parameterized Blocks

Different guidelines and rules apply depending on whether you want to use the parameterized block in more than one place in your program logic.

Parameterized block called from one block

If your parameterized block that contains a timer will be called from only one logic block, follow these rules:

1. Call the parameterized block exactly one time per execution of its source block.
2. Choose a reference address for the timer that will not be manipulated anywhere else. The reference address may be %R, %P, %L, %W, or symbolic.

Note: %L memory is the same %L memory available to the source block of type Block. %L memory corresponds to %P memory when the source block is _MAIN.

Parameterized block called from multiple blocks

When calling the parameterized block from multiple blocks, it is imperative to separate the timer reference memory used by each call to the parameterized block. Follow these rules and guidelines:

1. Call the parameterized block exactly one time per execution of each source block that it appears in.
2. Choose a %L reference or parameterized block formal parameter for the timer reference memory. Do not use a %R, %P, %W, or symbolic memory reference.

Notes:

- The strongly recommended choice is a %L location, which is inherited from the parameterized block's source block. Each source block has its own %L memory space except the _MAIN block, which has a %P memory area instead. When the _MAIN block calls another block, the %P mappings from the _MAIN block are accessed by the called block as %L mappings.
- If you use a parameterized block formal parameter (word array passed-by-reference), the actual parameter that corresponds to this formal parameter must be a %L, %R, %P, %W, or symbolic reference. If the actual parameter is a %R, %P, %W, or symbolic reference, a unique reference address must be used by each source block.

Recursion

If you use recursion (that is, if you have a block call itself either directly or indirectly) and your parameterized block contains an OFDT, ONDTR, or TMR, you must follow two additional rules:

- Program the source block so that it invokes the parameterized block before making any recursive calls to itself.
- Do not program the parameterized block to call itself directly.

Using OFDT, ONDTR and TMR Functions in UDFBs

UDFBs are user-defined logic blocks that have parameters and instance data. For details on these and other types of blocks, refer to Chapter 6.

When a timer function is present inside a UDFB, and a member variable is used for the control block of a timer, the behavior of the timer may not match your expectations. If multiple instances of the UDFB are called during a logic sweep, only the first-executed instance will update its timer. If a different instance is then executed, its timer value will remain unchanged.

In the case of multiple calls to a UDFB during a logic scan, only the first call will add elapsed time to its timer functions. This behavior matches the behavior of timers in a normal program block.

Example

A UDFB is defined that uses a member variable for a timer function block. Two instances of the function block are created: timer_A and timer_B. During each logic scan, both timer_A and timer_B are executed. However, only the member variable in timer_A is updated and the member variable in timer_B always remains at 0.

Off Delay Timer



Mnemonics:
OFDT_SEC
OFDT_TENTHS
OFDT_HUNDS
OFDT_THOUS

The Off-Delay Timer (OFDT) increments while power flow is off, and the timer's Current Value (CV) resets to zero when power flow is on. OFDT passes power until the specified interval PV (Preset Value) has elapsed.

Time may be counted in the following increments:

- Seconds
- Tenths (0.1) of a second
- Hundredths (0.01) of a second
- Thousandth (0.001) of a second

The range for PV is 0 to +32,767 time units. If PV is out of range, it has no effect on the timer's word 2. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

When OFDT receives power flow, CV is set to zero and the timer passes power to the right. The output remains on as long as OFDT receives power flow.

Each time the OFDT is invoked with its power flow input turned OFF, CV is updated to reflect the elapsed time since the timer was reset. OFDT continues passing power to the right until CV equals or exceeds PV. When this happens, OFDT stops passing power flow to the right and stops accumulating time. If PV is 0 or negative, the timer stops passing power flow to the right the first time that it is invoked with its power flow input OFF.

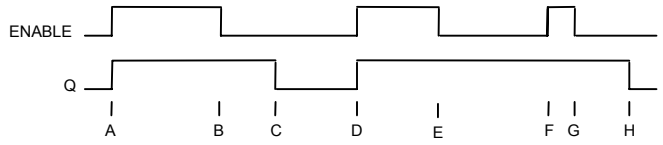
When the function receives power flow again, CV resets to zero.

Notes:

- The best way to use an OFDT function is to invoke it with a particular reference address exactly one time each scan. Do not invoke an OFDT with the same reference address more than once per scan (inappropriate accumulation of time would result). When an OFDT appears in a program block, it accumulates time once per scan. Subsequent calls to that program block within the same scan will have no effect on its OFDTs.
- Do not program an OFDT function with the same reference address in two different blocks. You should not program a JUMP around a timer function. Also, if you use recursion (that is. having a block call itself either directly or indirectly), program the program block so that it invokes the timer before it makes any recursive calls to itself.
- For information on using timers inside parameterized blocks, see page 8-141.
- An OFDT expires (turns OFF power flow to the right) the first scan that it does not receive power flow if the previous scan time was greater than PV.

- When OFDT is used in a program block that is not called every scan, the timer accumulates time between calls to the program block unless it is reset. This means that OFDT functions like a timer operating in a program with a much slower scan than the timer in the main program block. For program blocks that are inactive for a long time, OFDT should be programmed to allow for this catch-up feature. For example, if a timer in a program block is reset and the program block is not called (is inactive) for four minutes, when the program block is called, four minutes of time will already have accumulated. If the enable input is OFF, these four minutes are applied to the timer (that is, CV is set to 4 minutes).

Timing diagram



A. ENABLE and Q both go high; timer is reset (CV = 0).
B. ENABLE goes low; timer starts accumulating time.
C. CV reaches PV; Q goes low and timer stops accumulating time.
D. ENABLE goes high; timer is reset (CV = 0).
E. ENABLE goes low; timer starts accumulating time.
F. ENABLE goes high; timer is reset (CV = 0) before CV had a chance to reach PV. (The diagram is not to scale.)
G. ENABLE goes low; timer begins accumulating time.
H. CV reaches PV; Q goes low and timer stops accumulating time.

Operands for OFDT

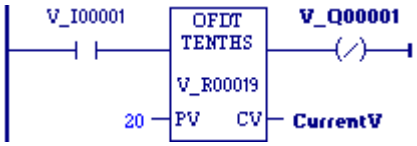
Warning

Do not use the Address, Address+1, or Address+2 addresses with other instructions. Overlapping references cause erratic timer operation.

Parameter	***Description***	***Allowed Operands***	***Optional***
Address (????)	The beginning address of a three-word WORD array: Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word	R, W, P, L, symbolic	No
PV	The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2.	All except S, SA, SB, SC	Optional
CV	The current value of the timer.	All except S, SA, SB, SC, constant	Optional

Example for OFDT

The output action is reversed by the use of a negated output coil. In this circuit, the OFDT timer turns off negated output coil %Q0001 whenever contact %I0001 is closed. After %I0001 opens, %Q0001 stays off for 2 seconds then turns on.



On Delay Stopwatch Timer



Mnemonics:
ONDTR_SEC
ONDTR_TENTHS
ONDTR_HUNDS
ONDTR_THOUS

The retentive On-Delay Stopwatch Timer (ONDTR) increments while it receives power flow and holds its value when power flow stops. Time may be counted in the following increments:

- Seconds
- Tenths (0.1) of a second
- Hundredths (0.01) of a second
- Thousandths (0.001) of a second

The range is 0 to +32,767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

When ONDTR first receives power flow, it starts accumulating time (Current Value (CV)). When the CV equals or exceeds Preset Value (PV), output Q is energized, regardless of the state of the power flow input.

As long as the timer continues to receive power flow, it continues accumulating until CV equals the maximum value (+32,767 time units). Once the maximum value is reached, it is retained and Q remains energized regardless of the state of the enable input.

When power flow to the timer stops, CV stops incrementing and is retained. Output Q, if energized, will remain energized. When ONDTR receives power flow again, CV again increments, beginning at the retained value.

When reset (R) receives power flow and PV is not equal to zero, CV is set back to zero and output Q is de-energized.

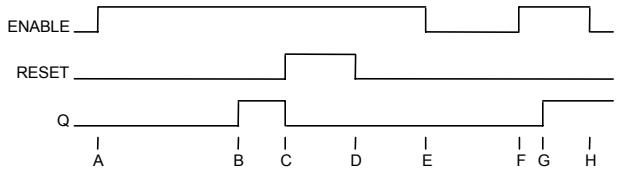
Note: If PV equals zero, the time is disabled and the reset is activated, and the output of the time becomes high. Subsequent removal of the reset or activation of input will have no effect on the timer output; the output of the time remains high.

ONDTR passes power flow to the right when CV is greater than or equal to PV. Since no automatic initialization to the outgoing power flow state occurs at power-up, the power flow state is retentive across power failure.

Notes:

- The best way to use an ONDTR function is to invoke it with a particular reference address exactly one time each scan. Do not invoke an ONDTR with the same reference address more than once per scan (inappropriate accumulation of time would result). When an ONDTR appears in a program block, it will only accumulate time once per scan. Subsequent calls to that same program block within the same scan will have no effect on its ONDTRs. Do not program an ONDTR function with the same reference address in two different blocks. You should not program a JUMPN around a timer function. Also, if you use recursion (that is, having a block call itself either directly or indirectly), program the program block so that it invokes the timer before it makes any recursive calls to itself.
- For information on using timers inside parameterized blocks, see page 8-141.
- An ONDTR expires (passes power flow to the right) the first scan that is enabled and not reset if the previous scan time was greater than PV.
- When ONDTR is used in a program block that is not called every scan, it accumulates time between calls to the program block unless it is reset. This means that ONDTR functions like a timer operating in a program with a much slower scan than the timer in the main program block. For program blocks that are inactive for a long time, ONDTR should be programmed to allow for this catch-up feature. For example, if a timer in a program block is reset and the program block is not called (is inactive) for four minutes, when the program block is called, four minutes of time will already have accumulated. If the enable input is ON and the reset input is OFF, these four minutes are applied to the timer (that is, CV is set to 4 minutes).

Timing diagram



A. ENABLE goes high; timer starts accumulating.
B. Current value (CV) reaches preset value (PV); Q goes high. Timer continues to accumulate time until ENABLE goes low, RESET goes high or current value becomes equal to the maximum time.
C. RESET goes high; Q goes low, accumulated time is reset (CV=0).
D. RESET goes low; timer then starts accumulating again, as ENABLE is high.
E. ENABLE goes low; timer stops accumulating. Accumulated time stays the same.
F. ENABLE goes high again; timer continues accumulating time.
G. CV becomes equal to PV; Q goes high. Timer continues to accumulate time until ENABLE goes low, RESET goes high or CV becomes equal to the maximum time.
H. ENABLE goes low; timer stops accumulating time.

Operands for On Delay Stopwatch Timer

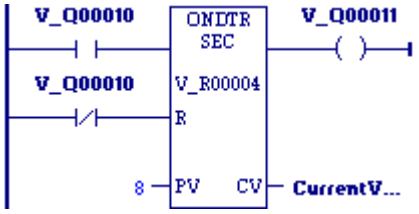
Warning

Do not use the Address, Address+1, or Address+2 addresses with other instructions. Overlapping references cause erratic timer operation.

Parameter	Description	Allowed Operands	Optional
Address (????)	Beginning address of a three-word WORD array: Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word	R, W, P, L, symbolic	No
R	When R is ON, it resets the Current Value (Word 1) to zero.	Power flow	Optional
PV	The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2.	All except S, SA, SB, SC	Optional
CV	Current Value of the timer	All except S, SA, SB, SC and constant	Optional

Example for On Delay Stopwatch Timer

A retentive on-delay timer is used to create a signal (%Q0011) that turns on 8.0 seconds after %Q0010 turns on, and turns off when %Q0010 turns off.



On Delay Timer



Mnemonics:

TMR_SEC
TMR_TENTHS
TMR_HUNDS
TMR_THOUS

The On-Delay Timer (TMR) increments while it receives power flow and resets to zero when power flow stops. The timer passes power after the specified interval PV (Preset Value) has elapsed, as long as power is received.

The range for PV is 0 to +32,767 time units. If PV is out of range, it has no effect on the timer's word 2. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

Time may be counted in the following increments:

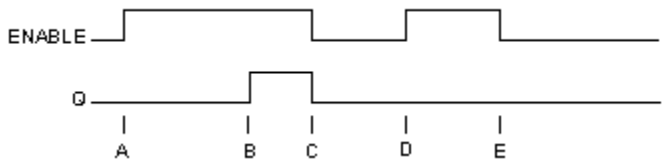
- Seconds
- Tenths (0.1) of a second
- Hundredths (0.01) of a second
- Thousandths (0.001) of a second

When TMR is invoked with its power flow input turned OFF, its Current Value (CV) is reset to zero, and the timer does not pass power flow to the right. Each time the TMR is invoked with its power flow input turned ON, CV is updated to reflect the elapsed time since the timer was reset. When CV reaches PV, the timer function passes power flow to the right.

Notes:

- The best way to use a TMR function is to invoke it with a particular reference address exactly one time each scan. Do not invoke a TMR with the same reference address more than once per scan (inappropriate accumulation of time would result). When a TMR appears in a program block, it will only accumulate time once per scan. Subsequent calls to that same program block within the same scan will have no effect on its TMRs. Do not program an TMR function with the same reference address in two different blocks. You should not program a JUMP around a timer function. Also, if you use recursion (that is, having a block call itself either directly or indirectly), program the program block so that it invokes the timer before it makes any recursive calls to itself.
- For information on using timers inside parameterized blocks, see page 8-141.
- A TMR timer expires (passes power flow to the right) the first scan that it is enabled if the previous scan time was greater than PV.
- When TMR is used in a program block that is not called every scan, TMR accumulates time between calls to the program block unless it is reset. This means that it functions like a timer operating in a program with a much slower sweep than the timer in the main program block. For program blocks that are inactive for a long time, TMR should be programmed to allow for this catch-up feature. For example, if a timer in a program block is reset and the program block is not called (is inactive) for 4 minutes, when the program block is called, four minutes of time will already have accumulated. If the enable input is ON, these four minutes are applied to the timer (i.e. CV is set to 4 minutes).

Timing Diagram



A. ENABLE goes high; timer begins accumulating time.

B. CV reaches PV; Q goes high and timer continues accumulating time.

C. ENABLE goes low; Q goes low; timer stops accumulating time and CV is cleared.

D. ENABLE goes high; timer starts accumulating time.

E. ENABLE goes low before current value reaches PV; Q remains low; timer stops accumulating time and is cleared to zero (CV=0).

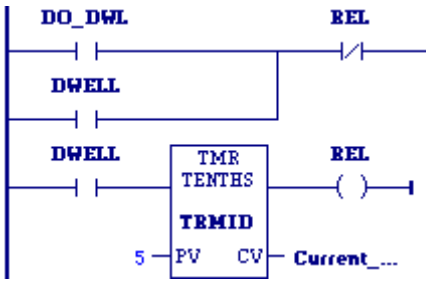
Operands for On Delay Timer

Warning

Do not use the Address, Address+1, or Address+2 addresses with other instructions. Overlapping references cause erratic timer operation.

Parameter	*Description*	*Allowed Operands*	*Optional*
????	The beginning address of a three-word WORD array: Word 1: Current value (CV) Word 2: Preset value (PV) Word 3: Control word	R, W, P, L, symbolic	No
PV	The Preset Value, used when the timer is enabled or reset. 0 ≤ PV ≤ +32,767. If PV is out of range, it has no effect on Word 2.	All except S, SA, SB, SC	Yes
CV	The current value of the timer.	All except S, SA, SB, SC and constant	Yes

Example for On Delay Timer



An on-delay timer with address TMRID is used to control the length of time that a coil is on. This coil has been assigned the variable DWELL. When normally open (momentary) contact DO_DWL is ON, coil DWELL is energized.

The contact of coil DWELL keeps coil DWELL energized (when contact DO_DWL is released) and also starts the timer TMRID. When TMRID reaches its preset value of five tenths of a second, coil REL energizes, interrupting the latched-on condition of coil DWELL. The contact DWELL interrupts power flow to TMRID, resetting its current value and de-energizing coil REL. The circuit is then ready for another momentary activation of contact DO_DWL.

Standard Timer Function Blocks

The standard timers are a pulse timer (TP), an on-delay timer (TON), and an off-delay timer (TOF). The pulse timer block can be used to generate output pulses of a given duration. The on-delay timer can be used to delay setting an output ON for a fixed period after an input is set ON. The off-delay timer can be used to delay setting an output OFF for a fixed period after an input goes OFF so that the output is held on for a given period longer than the input.

Notes:

- Any block type can contain calls to the standard timers. (See Chapter 6 for a discussion of the various block types.)
- Interrupt blocks can contain standard timers.
- An instance of a timer can be passed by reference to a parameterized block or UDFB.
- When the timer stops timing as a result of reaching its Preset Time (PT), the Elapsed Time (ET) contains the actual timer duration. For example, if the Preset Time was specified as 333 ms, but the timer actually timed to 350 ms, the 350 ms value is saved in ET.

Data Required for Standard Timer Function Blocks

Each invocation of a timer has associated instance data that persists from one execution of the timer to the next. Instance variables are automatically located in symbolic memory. (You cannot specify an address.) You can specify a stored value for each element. The user logic cannot modify the values.

Each timer instance variable has the following structure. Elements of a timer structure cannot be published.

The instance data type for each timer must be the same as the timer type:

The TOF timer requires an instance variable of type TOF.
The TON timer requires an instance variable of type TON.
The TP timer requires an instance variable of type TP.

Element	*Type*	*Description*	*Details*
IN	BOOL	Timer input	Cannot be accessed in user logic.
PT	DINT	Preset time	Cannot be accessed in user logic.
ET	DINT	Elapsed time	Read only. Accessible in user logic.
Q	BOOL	Set ON when timer finishes timing	Read only. Accessible in user logic.
ENO	BOOL	Enable output	Read only. Accessible in user logic.
TI	BOOL	Set ON when the timer instance is timing (that is, ET is incrementing).	Read only. Accessible in user logic.

Resetting the Timer

The preset time (PT) may be changed while the timer is timing to affect the duration.

When the timer reaches PT, the timer stops timing and the elapsed time parameter (ET) contains the actual timer duration.

To reset the timer function block, set the PT input to 0. When the function block resets:

ET is set to 0
Q is set to off (0)
The TI element is set to 0
The IN parameter is ignored

Operands

TOF, TON and TP have the same input and output parameters, except for the instance variable, which must be the same type as the instruction.

Note: Writing or forcing values to the instance data elements IN, PT, Q, ET, ENO or TI may cause erratic operation of the timer function block.

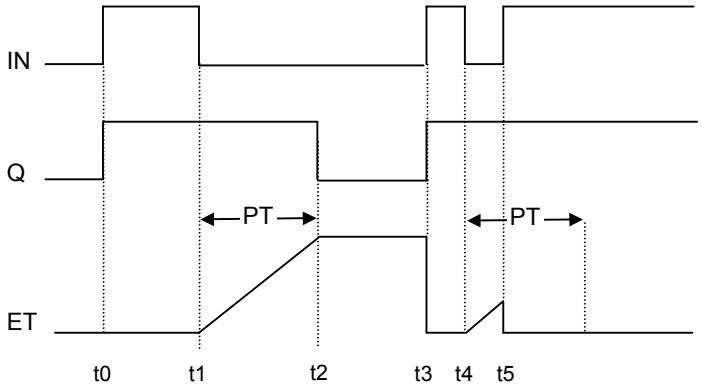
Parameter	Description	Allowed Types	Allowed Operands	Optional
????	Structure variable containing the internal data for the timer instance. (See "Data Required for Standard Timer Function Blocks" on page 8-151.)	TOF, TON, or TP. Must be same type as the instruction.	NA	No
IN	Timer input. Controls when the timer will accumulate time. TON and TP will begin to time when IN transitions from OFF to ON. TOF will begin to time when IN transitions from ON to OFF.	Flow	NA	Yes
PT	Preset time (in milliseconds). Indicates the amount of time the timer will time until turning Q either ON or OFF, depending on the timer type. Setting PT to 0 resets the timer.	DINT	All except S, SA, SB, SC	Yes
Q	Timer output. Action depends on the timer type. When TP is timing, Q is ON. When TON is done timing, Q turns ON. When TOF is done timing, Q turns OFF.	Flow	NA	Yes
ET	Elapsed time. Indicates the length of time, in milliseconds, that the timer has been measuring time.	DINT	All except S, SA, SB, SC and constants	Yes

Timer Off Delay



When the input IN transitions from ON to OFF, the timer starts timing until a specified period of time (PT) has elapsed, then sets the output Q to OFF.

Timing Diagram



t0 When input IN is set to ON, the output Q follows and remains ON. The elapsed time, ET, does not increment.

t1 When IN goes OFF, the timer starts to measure time and ET increments. ET continues to increment until its value equals the preset time, PT.

t2 When ET equals PT, Q is set to OFF and ET remains at the preset time, PT.

t3 When input IN is set to ON, the output Q follows and remains ON. ET is set to 0.

t4 When IN is set to OFF, ET, begins incrementing. When IN is OFF for a period shorter than that specified by PT, Q remains ON.

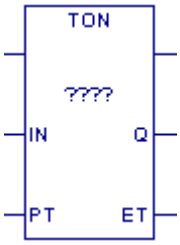
t5 When IN is set to ON, ET is set to 0.

Example

In the following sample rung, a TOF function block is used to keep Light ON for 30,000 ms (30 seconds) after Door_Open is set to OFF. As long as Door_Open is ON, Light remains ON.

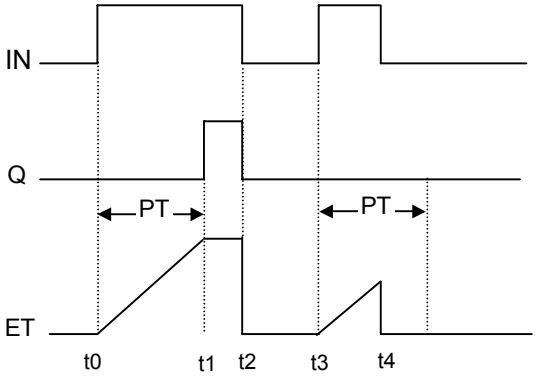


Timer On Delay



When the input IN transitions from OFF to ON, the timer starts timing until a specified period of time (PT) has elapsed, then sets the output Q to ON.

Timing Diagram



t0 When input IN is set to ON, the timer starts to measure time and the elapsed time output ET starts to increment. The output Q remains OFF and ET continues to increment until its value equals the preset time, PT.

t1 When ET equals PT, the output Q is goes ON, and ET remains at the preset time, PT. Q remains ON until IN goes OFF.

t2 When IN is set to OFF, Q goes OFF and ET is set to 0.

t3 When IN is set to ON, ET starts to increment.

t4 If IN is ON for a shorter time than the delay specified in PT, the output Q remains OFF. ET is set to 0 when IN is set to OFF.

Example

In the following sample rung, a TON function block is used to delay setting Start to ON for 1 minute (60,000 ms) after Preheat is set to ON.

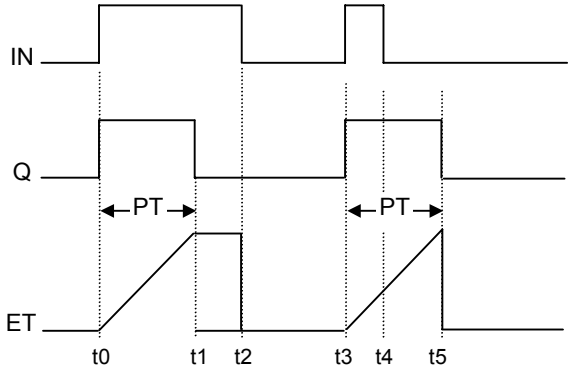


Timer Pulse



When the input IN transitions from OFF to ON, the timer sets the output Q to ON for the specified time interval, PT

Timing Diagram



t0 When input IN is set to ON, the timer starts to measure time and the elapsed time output, ET, increments until its value equals that of the specified preset time, PT. Q is set to 0 on until ET equals PT.

t1 When ET equals PT, Q is set to OFF. The value of ET is held until IN is set to OFF.

t2 When IN is set to OFF, ET is set to 0.

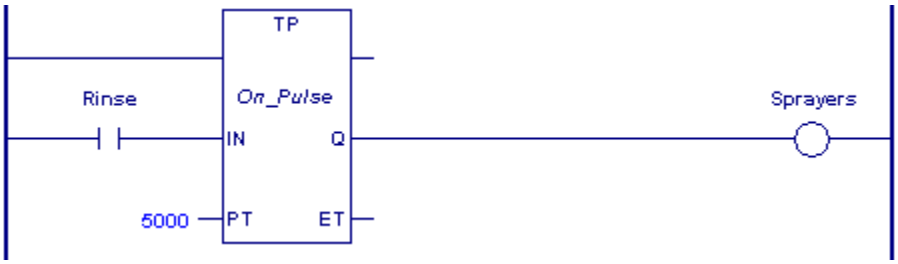
t3 When IN is set to ON, the timer starts to measure time and ET begins incrementing. Q is set to ON.

t4 If the input is OFF for a period shorter than the input PT, the output Q remains on and ET continues incrementing.

t5 When ET equals PT, Q is set to OFF and ET is set to 0.

Example

In the following sample rung, a TP function block is used to set Sprayers to ON for a 5-second (5000 ms) pulse.



Chapter 9

Function Block Diagram

Function Block Diagram (FBD) is an IEC 61131-3 graphical programming language that represents the behavior of functions, function blocks and programs as a set of interconnected graphical blocks.

The block types Block, Parameterized Block, and Function Block can be programmed in FBD. The _MAIN program block can also be programmed in FBD. For details on blocks, refer to chapter 6, “Program Organization.” For information on using the FBD editor in the programming software, refer to the online help.

For an overview of the types of operands that can be used with instructions, refer to “Operands for Instructions” in chapter 7.

Most functions and function blocks implemented in FBD are the same as their LD counterparts. Instructions that are implemented differently are discussed in detail in this chapter.

FBD has the following general differences compared to LD:

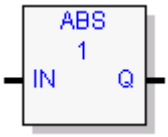
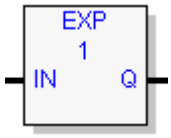
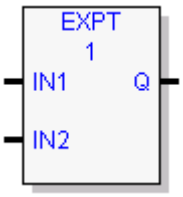
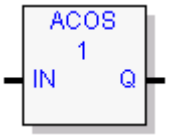
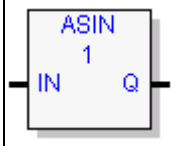
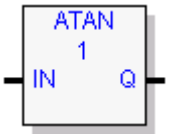
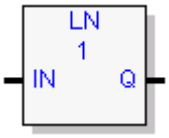
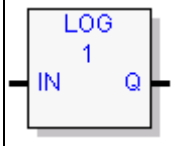
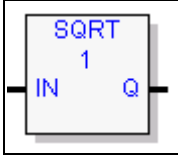
- In FBD, except for timers and counters, functions and function blocks do not have EN or ENO parameters.
- In FBD, all functions and function blocks display a solve order, which is calculated by the FBD editor.

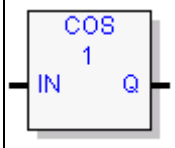
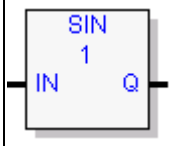
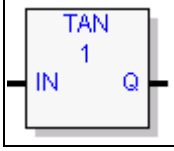
The FBD implementation of the PACSystems instruction set includes the following categories:

- Advanced Math 9-2
- Bit Operations 9-4
- Comment Block 9-9
- Comparison Functions 9-10
- Control Functions 9-13
- Counters 9-15
- Data Move Functions 9-16
- Math Functions 9-21
- Program Flow Functions 9-29
- Timers 9-30
- Type Conversion Functions 9-32

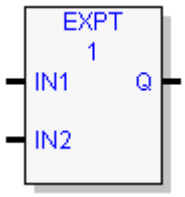
Advanced Math Functions

The Advanced Math functions perform logarithmic, exponential, square root, trigonometric, and inverse trigonometric operations.

Function	Description
	Absolute value. Finds the absolute value of a double- precision integer (DINT), signed single-precision integer (INT), REAL or LREAL (floating-point) value. The mnemonic specifies the value's data type. For details, see “Math Functions” in chapter 8.
	Exponential. Raises **e** to the value specified in IN ($\mathbf{e}^{IN}$). Calculates the inverse natural logarithm of the IN operand. For details, see “Advanced Math Functions” in chapter 8.
	Exponential. Calculates IN1 to the IN2 power ($IN1^{IN2}$). For details, see page 9-3.
	Inverse trig. Calculates the inverse cosine of the IN operand and expresses the result in radians. For details, see “Advanced Math Functions” in chapter 8.
	Inverse trig. Calculates the inverse sine of the IN operand and expresses the result in radians. For details, see “Advanced Math Functions” in chapter 8.
	Inverse trig. Calculates the inverse tangent of the IN operand and expresses the result in radians. For details, see “Advanced Math Functions” in chapter 8.
	Logarithmic. Calculates the natural logarithm of the operand IN. For details, see “Advanced Math Functions” in chapter 8.
	Logarithmic. Calculates the base 10 logarithm of the operand IN. For details, see “Advanced Math Functions” in chapter 8.
	Square root. Calculates the square root of the operand IN and stores the result in Q. For details, see “Advanced Math Functions” in chapter 8.

Function	Description
	Trig. Calculates the cosine of the operand IN, where IN is expressed in radians. For details, see “Advanced Math Functions” in chapter 8.
	Calculates the sine of the operand IN, where IN is expressed in radians. For details, see “Advanced Math Functions” in chapter 8.
	Calculates the tangent of the operand IN, where IN is expressed in radians. For details, see “Advanced Math Functions” in chapter 8.

EXPT Function



The Power of X (EXPT) function raises the value of input IN1 to the power specified by the value IN2 and places the result in Q. The EXPT function operates on REAL or LREAL input value(s) and place the result in output Q. The instruction is not carried out if one of the following invalid conditions occurs:

- IN1 < 0, for EXPT
- IN1 or IN2 is a NaN (Not a Number)

Invalid operations (error cases) may yield results that are different from those in the LD implementation of this function.

Operands of the EXPT Function

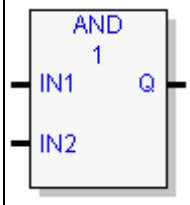
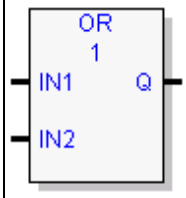
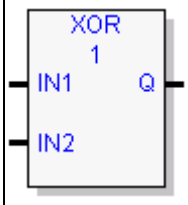
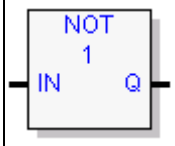
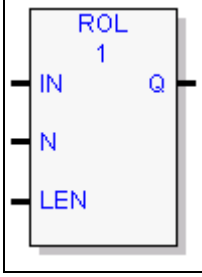
Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN or IN1	For EXP, LOG, and LN, IN contains the REAL value to be operated on. The EXPT function has two inputs, IN1 and IN2. For EXPT, IN1 is the base value and IN2 is the exponent.	REAL, LREAL	All except variables located in %S—%SC	No
IN2 (EXPT)	The REAL exponent for EXPT.	REAL, LREAL	All except variables located in %S—%SC	No
Q	Contains the REAL logarithmic/exponential value of IN or of IN1 and IN2.	REAL, LREAL	All except constants and variables located in %S—%SC	No

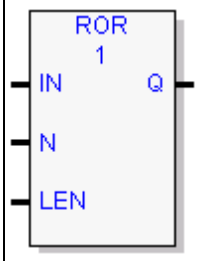
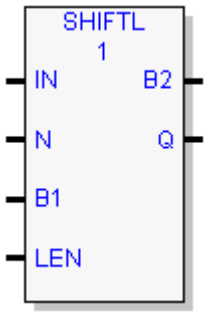
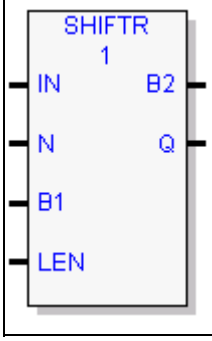
Bit Operation Functions

The Bit Operation functions perform comparison, logical, and move operations on bit strings. Bit Operation functions treat each WORD or DWORD data as a continuous string of bits, with bit 1 of the WORD or DWORD being the Least Significant Bit (LSB). The last bit of the WORD or DWORD is the Most Significant Bit (MSB).

Warning

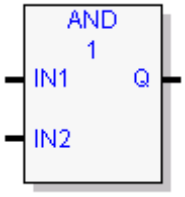
Overlapping input and output reference address ranges in multiword functions is not recommended, as it can produce unexpected results.

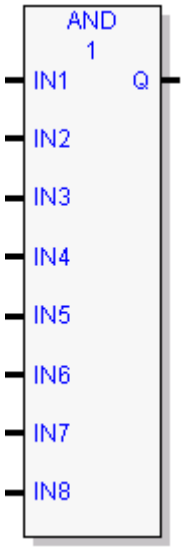
Function	*Description*
	Logical AND. Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are both 1, places a 1 in the corresponding location in output string Q; otherwise, places a 0 in the corresponding location in Q. If additional inputs (IN3 through IN8) are used, each additional bit string is compared to the string in Q and the result is placed in Q. For details, see page 9-6.
	Logical OR. Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are both 0, places a 0 in the corresponding location in output string Q; otherwise, places a 1 in the corresponding location in Q. If additional inputs (IN3 through IN8) are used, each additional bit string is compared to the string in Q and the result is placed in Q. For details, see page 9-6.
	Logical XOR. Compares the bit strings IN1 and IN2 bit by bit. When a pair of corresponding bits are different, places a 1 in the corresponding location in the output bit string Q; when a pair of corresponding bits are the same, places a 0 in Q. If additional inputs (IN3 through IN8) are used, each additional bit string is compared to the string in Q and the result is placed in Q. For details, see page 9-6.
	Logical NOT. Sets the state of each bit in output bit string Q to the opposite state of the corresponding bit in bit string IN1. For details, see page 9-8.
	Rotate Bits Left. Rotates all the bits in a string a specified number of places to the left. For details, see “Bit Operation Functions” in chapter 8.

Function	Description
	Rotate Bits Right. Rotates all the bits in a string a specified number of places to the right. For details, see “Bit Operation Functions” in chapter 8.
	Shift Bits Left. Shifts all the bits in a word or string of words to the left by a specified number of places. For details, see “Bit Operation Functions” in chapter 8.
	Shift Bits Right. Shifts all the bits in a word or string of words to the right by a specified number of places. For details, see “Bit Operation Functions” in chapter 8.

Logical AND, Logical OR, and Logical XOR

The Logical functions examine each bit in bit string IN1 and the corresponding bit in bit string IN2, beginning with the least significant bit in each string, and places the result in Q. If additional inputs (IN3 up to IN8) are used, the function compares each bit in the input with the corresponding bit in Q and places the result in Q. The comparison is repeated for each input that is used. The input bit strings specified in IN1 … IN8 may overlap.



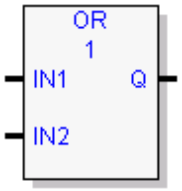


Logical AND

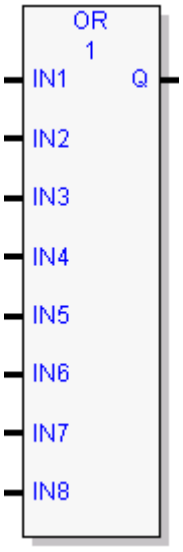
If both bits examined by the Logical AND function are 1, AND places a 1 in the corresponding location in output string Q. If either bit is 0 or both bits are 0, AND places a 0 in string Q in that location.

Tip: You can use the Logical AND function to build masks or screens, where only certain bits are passed (the bits opposite a 1 in the mask), and all other bits are set to 0.

Minimum number of inputs = 2
Maximum number of inputs = 8



Minimum number of inputs = 2



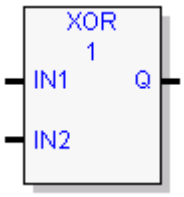
Maximum number of inputs = 8

Logical OR

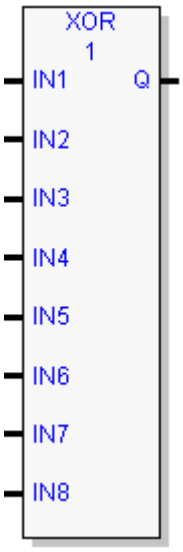
If either bit examined by the Logical OR function is 1, OR places a 1 in the corresponding location in output string Q. If both bits are 0, Logical OR places a 0 in string Q in that location.

Tips:

- You can use the Logical OR function to combine strings or to control many outputs with one simple logical structure. The Logical OR function is the equivalent of two relay contacts in parallel multiplied by the number of bits in the string.
- You can use the Logical OR function to drive indicator lamps directly from input states or to superimpose blinking conditions on status lights.



Minimum number of inputs = 2



Maximum number of inputs = 8

Logical XOR

If the bits in the strings examined by XOR are different, a 1 is placed in the corresponding position in the output bit string.

For each pair of bits examined, if only one bit is 1, XOR places a 1 in the corresponding location in string Q.

If both bits are 0, XOR places a 0 in the corresponding location in string Q.

Tips:

- If string IN2 and output string Q begin at the same reference, a 1 placed in string IN1 will cause the corresponding bit in string IN2 to alternate between 0 and 1, changing state with each scan as long as input is received.
- You can program longer cycles by pulsing the input to the function at twice the desired rate of flashing. The input pulse should be one scan long (one-shot type coil or self resetting timer).
- You can use XOR to quickly compare two bit strings, or to blink a group of bits at the rate of one ON state per two scans.
- XOR is useful for transparency masks.

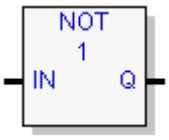
Operands for AND, OR, and XOR

Parameter	***Description***	***Allowed Types***	***Allowed Operands***	***Optional***
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1	The value to operate on.	BOOL, WORD DWORD	All	No
IN2 (Must be the same data type as IN1.)	The value to operate on.	BOOL, WORD DWORD	All	No
IN3 … IN8 (Must be the same data type as IN1.)	Values to operate on.	BOOL, WORD DWORD	All	Yes
Q (Must be the same data type as IN1 and IN2.)	The operation's result.	BOOL, WORD DWORD	All except constants and variables located in %S memory	No

Properties for AND, OR, and XOR

Property	***Valid Range***
Number of Inputs	2 to 8

Logical NOT



The Logical Not or Logical Invert (NOT) function sets the state of each bit in the output bit string Q to the opposite of the state of the corresponding bit in bit string IN1.

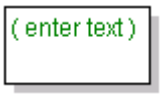
All bits are altered on each scan that input is received, making output string Q the logical complement of input string IN1.

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1	The input string to NOT.	WORD DWORD	All	No
Q	The NOT's result.	WORD DWORD (Must be the same data type as IN1)	All except constants and variables located in %S memory	No

Comments

Text Block



The Text block is used to place an explanation in the program. When you type in a comment, the first few words are displayed.



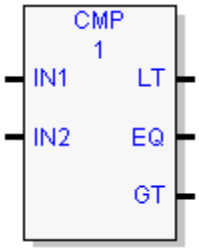
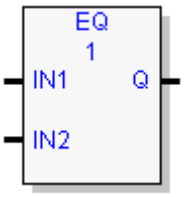
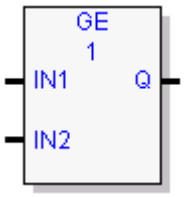
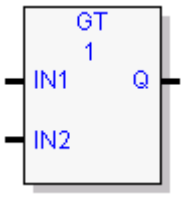
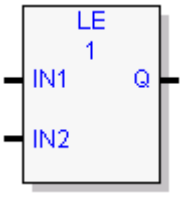
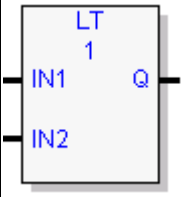
To increase the size of the text box and display more text, select the box and drag one of the handles.

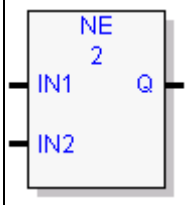
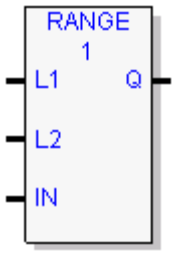
There are no operands for the Text block.

Note: Since comments are not downloaded to the PLC, you can edit them, whether offline or online, without losing equality.

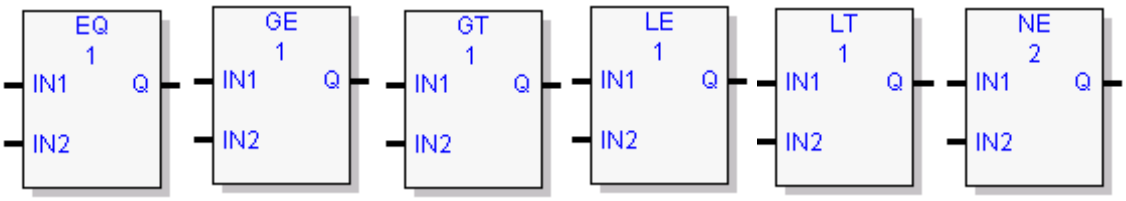
Comparison Functions

Comparison functions compare two values of the same data type or determine whether a number lies within a specified range. The original values are unaffected.

Function	Description
	Compare. Compares two numbers, IN1 and IN2. For details, see “Relational Functions” in chapter 8.
	Equal. Tests two numbers for equality. For details, see page 9-12.
	Greater Than or Equal. Tests whether one number is greater than or equal to another. For details, see page 9-12.
	Greater Than. Tests whether one number is greater than another. For details, see page 9-12.
	Less Than or Equal. Tests whether one number is less than or equal to another. For details, see page 9-12.
	Less Than. Tests whether one number is less than another. For details, see page 9-12.

Function	Description
	Not Equal. Tests whether two numbers are not equal. For details, see page 9-12.
	Range. Tests whether one number is within the range defined by two other supplied numbers. For details, see “Relational Functions” in chapter 8.

Equal, Not Equal, Greater or Equal, Greater Than, Less or Equal, Less Than



The relational functions compare input IN1 to input IN2. These operands must be the same data type. If inputs IN1 and IN2 are equal, the function outputs the result to Q, unless IN1 and/or IN2 is NaN (Not a Number). The following relational functions can be used to compare two numbers:

Function	*Definition*	*Relational Statement*
EQ	Equal	IN1=IN2
NE	Not Equal	IN1≠IN2
GE	Greater Than or Equal	IN1≥IN2
GT	Greater Than	IN1>IN2
LE	Less Than or Equal	IN1≤IN2
LT	Less Than	IN1<IN2

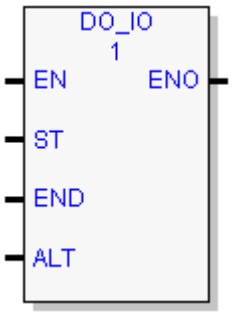
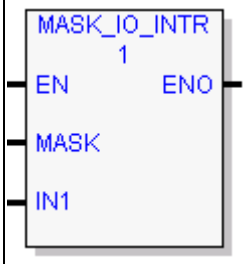
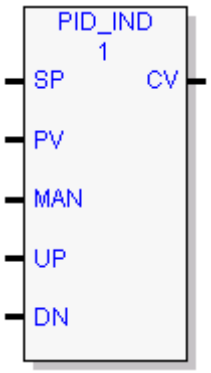
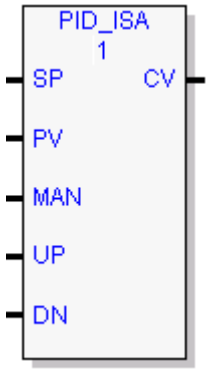
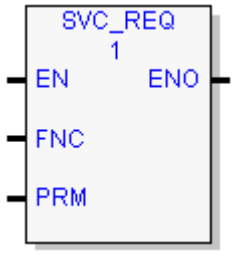
Tip: To compare values of different data types, first use conversion functions to make the types the same.

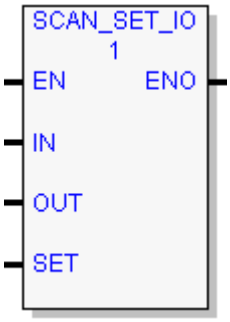
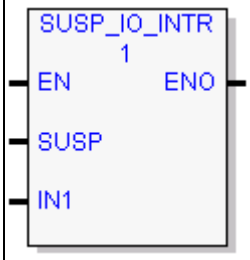
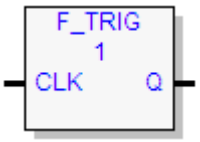
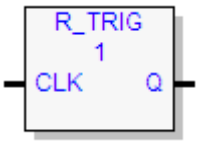
Operands

Parameter	*Description*	*Allowed Types*	*Allowed Operands*	*Optional*
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1	The first value to be compared; the value on the left side of the relational statement.	BOOL (for EQ and NE functions only), BYTE, DINT, DWORD, INT, REAL, LREAL, UINT, WORD	All except S, SA, SB, SC	No
IN2	The second value to be compared; the value on the right side of the relational statement. IN2 must be the same data type as IN1.			No
Q	If the relational statement is true, Q=1.	BOOL	I, Q, G, M, T, SA, SB, SC	No
		Bit reference in a non-BOOL variable.	All except constants.	

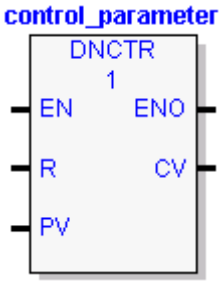
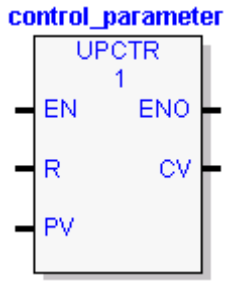
Control Functions

The control functions limit program execution and change the way the CPU executes the application program.

Function	Description
	Do I/O Interrupt. For one scan, immediately services a specified range of inputs or outputs. (All inputs or outputs on a module are serviced if any reference locations on that module are included in the DO I/O function. Partial I/O module updates are not performed.) Optionally, a copy of the scanned I/O can be placed in internal memory, rather than at the real input points. For details, see “Control Functions” in chapter 8.
	Mask I/O Interrupt. Mask or unmask an interrupt from an I/O board when using I/O variables. If not using I/O variables, use SVC_REQ 17, described in Chapter 10. For details, see “Control Functions” in chapter 8.
 	Proportional Integral Derivative (PID) Control. Provides two PID closed-loop control algorithms: Standard ISA PID algorithm (PID_ISA) Independent term algorithm (PID_IND) **Note:** For details, refer to chapter 11.
	Service Request. Requests a special PLC service. **Note:** For details, refer to chapter 10.

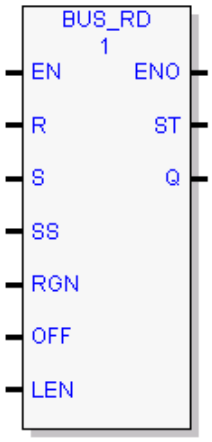
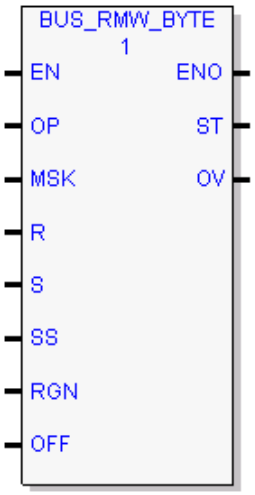
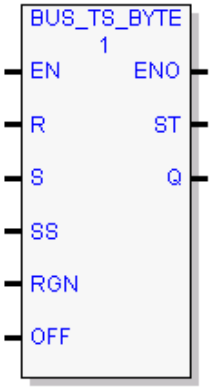
Function	Description
	Scan Set I/O. Scans the IO of a specified scan set. For details, see “Control Functions” in chapter 8.
	Suspend I/O. Suspends for one sweep all normal I/O updates, except those specified by DO I/O instructions. For details, see “Control Functions” in chapter 8.
	Suspend I/O Interrupt. Suspend or resume an I/O interrupt when using I/O variables. If not using I/O variables, use SVC_REQ 32, described in Chapter 10. For details, see “Control Functions” in chapter 8.
	Falling Edge Trigger. Detects a high-to-low transition of a Boolean input. Produces a single output pulse when a falling edge is detected. For details, see “Control Functions” in chapter 8.
	Rising Edge Trigger. Detects a low-to-high transition of a Boolean input. Produces a single output pulse when a rising edge is detected. For details, see “Control Functions” in chapter 8.

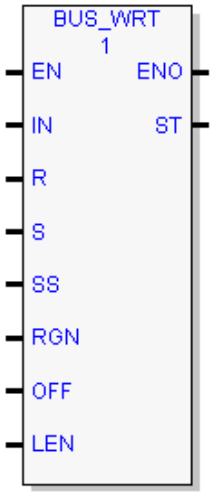
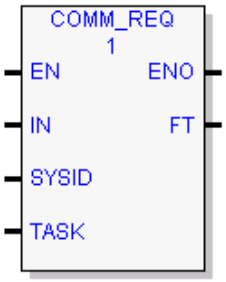
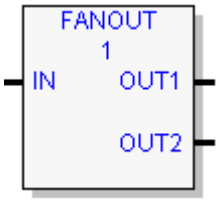
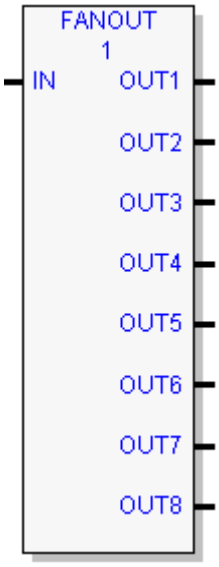
Counters

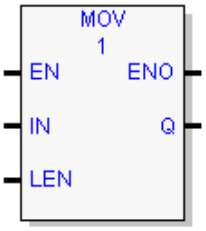
Function	Description
	Down Counter. Counts down from a preset value. The output is ON whenever the Current Value is ≤ 0. The parameter that appears above the function block is a one-dimensional, three-word array in %R, %W, %P, %L, or symbolic memory that the counter uses to store its current value, preset value and control word. For details, see “Counters” in chapter 8.
	Up Counter. Counts up to a designated value. The output is ON whenever the Current Value is ≥ the Preset Value. The parameter that appears above the function block is a one-dimensional, three-word array in %R, %W, %P, %L, or symbolic memory that the counter uses to store its current value, preset value and control word. For details, see “Counters” in chapter 8.

Data Move Functions

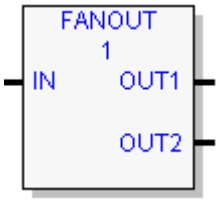
The Data Move functions provide basic data move capabilities.

Function	Description
	Bus Read. Reads data from the bus. For details, see “Data Move Functions” in chapter 8.
	Bus Read Modify Write. Uses a read/modify/write cycle to update a data element in a module on the bus. Other BUS_RMW functions: BUS_RMW_DWORD BUS_RMW_WORD For details, see “Data Move Functions” in chapter 8.
	Bus Test and Set. Handles semaphores on the bus. Other BUS_TS function: BUS_TS_WORD For details, see “Data Move Functions” in chapter 8.

Function	Description
	Bus Write. Writes data to a module on the bus. For details, see “Data Move Functions” in chapter 8.
	Communication Request. Allows the program to communicate with an intelligent module, such as a Genius Bus Controller or a High Speed Counter. For details, see “Communication Request” in chapter 8.
 Minimum Outputs = 2	 Maximum Outputs = 8	Fan Out. Copies the input value to multiple outputs of the same data type as the input. For details, see page 9-18.

Function	Description
	Move Data. Copies data as individual bits, so the new location does not have to be the same data type. Data can be moved into a different data type without prior conversion. For details, see page 9-18.

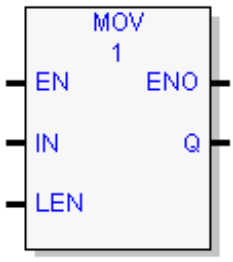
Fan Out



Copies the input IN to multiple outputs.

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN	The input to copy to the outputs.	BOOL, DINT, DWORD, INT, REAL, UINT, or WORD variable or constant	All except SA, SB, SC.	No
OUT1 …OUT8	Variables of the same data type as the IN operand. The outputs. Minimum: two outputs. Maximum: eight outputs.	Must be same type as IN.	All except S, SA, SB, SC and constant.	No

Move Data



When the input operand, EN, is set to ON, the MOVE instruction copies data as bits from one location in PLC memory to another. Because the data is copied as bits, the new location does not need to use the same type of memory area as the source. For example, you can copy data from an analog memory area into a discrete memory area, or vice versa.

MOV sets its output, ENO, whenever it receives data unless one of the following occurs:

- When the input, EN, is set to OFF, then the output, ENO, is set to OFF.
- When the input, EN is set to ON, and the input, IN, contains an indirect reference, and the memory of IN is out of range, then the output, ENO, is set to OFF.

The value to store at the destination Q is acquired from the IN parameter. If IN is a variable, the value to store in Q is the value stored at the IN address. If IN is a constant, the value to store in Q is that constant

The result of the MOVE depends on whether the data type for the Q operand is a bit reference or a non-bit reference:

- If Q is a non-bit reference, LEN (the length) indicates the number of memory locations in which the IN value should be repeated, starting at the location specified by Q.
- If Q is a bit reference, IN is treated as an array of bits. LEN therefore indicates the number of bits to acquire from the IN parameter to make up the stored value. If IN is a constant, bits are counted from the least-significant bit. If IN is a variable, LEN indicates the number of bits to acquire starting at the IN location. Regardless, only LEN bits are stored starting at address Q.

For example, if IN was the constant value 29 and LEN is 4, the results of a MOV operation are as follows:

- Q is a WORD reference: The value 29 is repeatedly stored in locations Q, Q+1, Q+2, and Q+3.
- Q is a BOOL reference: The binary representation of 29 is 11101. Since LEN is 4, only the four least-significant bits are used (1101). This value is stored at location Q in the same order, so 1 is stored in Q, 1 is stored in Q+1, 0 is stored in Q+2, and 1 is stored in Q+3.

If data is moved from one location in discrete memory to another, such as from %I memory to %T memory, the transition information associated with the discrete memory elements is updated to indicate whether or not the MOVE operation caused any discrete memory elements to change state.

Note: If an array of BOOL-type data specified in the Q operand does not include all the bits in a byte, the transition bits associated with that byte (which are not in the array) are cleared when the Move instruction receives data.

Data at the IN operand does not change unless there is an overlap in the source and destination—a situation that is to be avoided.

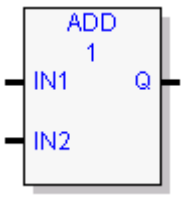
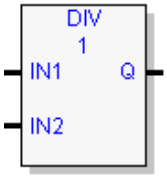
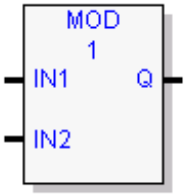
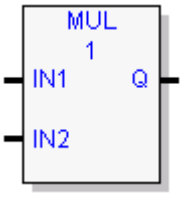
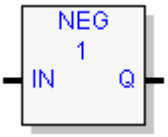
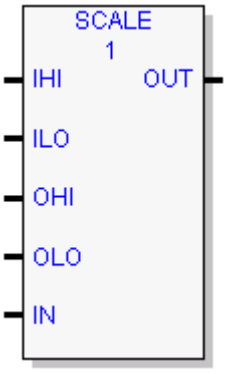
MOV Operands

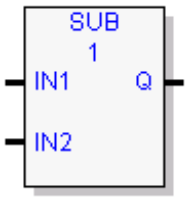
Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
EN	Enable	BOOL variable	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	No
		Bit reference in a non-BOOL variable	R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

Parameter	Description	Allowed Types	Allowed Operands	Optional
IN	The source of the data to copy into the output Q. This can be either a constant or a variable whose reference address is the location of the first source data item to move. If IN is not a constant, it must have the same data type as the variable in the Q parameter. The length of the data unit depends on the data type of the IN or Q variable. If IN is a BOOL variable or a bit reference, an %I, %Q, %M, or %T reference address need not be byte-aligned, but 16 bits beginning with the reference address specified are displayed online.	DINT, DWORD, INT, REAL, LREAL, UINT, WORD, or bit reference in a non-BOOL variable	All. S, SA, SB, SC allowed only for WORD, DWORD, BOOL types.	No
LEN	The length of IN; the number of bits to move. If IN is a constant and Q is BOOL: $1 \leq LEN \leq 16$; If IN is a constant and Q is *not* BOOL: $1 \leq LEN \leq 256$. All other cases: $1 \leq LEN \leq 32{,}767$ LEN is also interpreted differently depending on the data type of the Q location. For details, see discussion on page 9-18.	Constant	Constant	No
ENO	Indicates whether the operation was successfully completed. If ENO = ON (1), the operation was initiated. Results of the operation are indicated in the FT output. If ENO = OFF (0), the operation was not performed. If EN was ON, the FT output indicates an error condition. If EN was OFF, FT is not changed.	BOOL variable	data flow, I, Q, M, T, G, discrete symbolic, I/O variable	Yes
		Bit reference in a non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
Q	The variable whose reference address is the location of the first destination data item. If IN is not a constant, Q must have the same data type as the variable in the IN parameter. If Q is a BOOL variable or a bit reference, an %I, %Q, %M, or %T reference address does not need to be byte-aligned, but a full 16 bits beginning with the specified reference address are displayed online.	DINT, DWORD, INT, REAL, LREAL, UINT, WORD, or bit reference in a non-BOOL variable	data flow, I, Q, M, T, S, SA, SB, SC, G, R, P, L, AI, AQ, W, symbolic, I/O variable	No

Math Functions

Your program may need to include logic to convert data to a different type before using a Math or Numerical function. The description of each function includes information about appropriate data types. The “Conversion Functions” section on page 9-32 explains how to convert data to a different type.

Function	Description
	Addition. Adds two or up to eight numbers. For details, see page 9-23.
	Division. * Divides one number by another and outputs the quotient. **Note:** Take care to avoid overflow conditions when performing divisions. For details, see page 9-24.
	Modulo Division. Divides one number by another and outputs the remainder. For details, see page 9-25.
	Multiplication.* Multiplies two or up to eight numbers. **Note:** Take care to avoid overflow conditions when performing multiplications. For details, see page 9-26.
	Negate. Multiplies a number by –1 and places the result in an output location. For details, see page 9-27.
	Scales an input parameter and places the result in an output location. For details, see “Math Functions” in chapter 8.

Function	Description
	Subtraction. Subtracts one or up to seven numbers from the input IN1 and places the result in an output location. For details, see page 9-28.

* To avoid overflows when multiplying or dividing 16-bit numbers, use the conversion functions described on page 9-32 to convert the numbers to a 32-bit format.

The output is calculated when the instruction is performed without overflow, unless an invalid operation occurs.

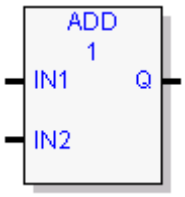
Overflow

If an operation on integer operands results in overflow, the output value wraps around.

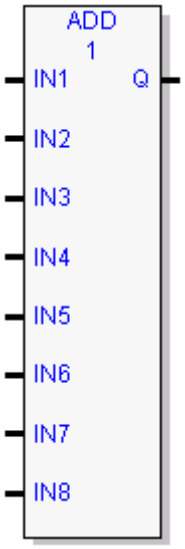
Examples:

- If the ADD operation, 32767 + 1, is performed on signed integer operands, the result is -32768
- If the SUB operation, -32767 – 1, is performed on signed integer operands, the result is 32767
- If an ADD_UINT operation is performed on 65535 + 16, the result is 15.

Add



Minimum number of inputs = 2



Maximum number of inputs = 8.

Adds the operands IN1 and IN2 … IN8 and stores the sum in Q. IN1 … IN8 and Q must be of the same data type.

The result is output to Q when ADD is performed without overflow, unless one of the following invalid conditions occurs:

- (+ ∞)
- IN1 and/or IN2 … IN8 is NaN (Not a Number).

If an ADD operation results in overflow, the result wraps around. For example:

- If an ADD_DINT, ADD_INT or ADD_REAL operation is performed on 32767 + 1, Q will be set to -32768.
- If an ADD_UINT operation is performed on 65535 + 16, Q will be set to 15.

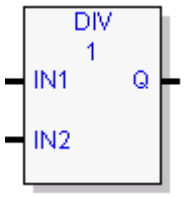
Operands of the ADD Function

Operand	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1 … IN8	The values to be added.	INT, DINT, REAL, LREAL, UINT Must be same data type as Q.	All except S, SA, SB, SC and data flow	No
Q	The sum of IN1 … IN8. If an overflow occurs, Q wraps around.	INT, DINT, REAL, LREAL, UINT variable Must be same data type as IN1 …. IN8.	All except S, SA, SB, SC, constant and data flow	No

Properties for ADD

Property	*Valid Range*
Number of Inputs	2 to 8

Divide



Divides the operand IN1 by the operand IN2 of the same data type as IN1 and stores the quotient in the output variable assigned to Q, also of the same data type as IN1 and IN2.

The result is output to Q when DIV is performed without overflow, unless one of the following invalid conditions occurs:

- 0 divided by 0 (Results in an application fault.)
- IN1 and/or IN2 is NaN (Not a Number).

If an overflow occurs, the result wraps around.

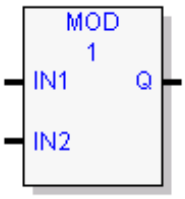
Notes:

- DIV rounds down; it does not round to the closest integer. For example, 24 DIV 5 = 4.
- Be careful to avoid overflows.

Operands for DIV_UINT, DIV_INT, DIV_DINT, and DIV_REAL

Parameter	*Description*	*Allowed Types*	*Allowed Operands*	*Optional*
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1	The value to be divided; the value to the left of “DIV” in the equation IN1 DIV IN2=Q.	INT, DINT, UINT, REAL, LREAL	All except S, SA, SB, SC	No
IN2	The value to divide IN1 with; the value to the right of “DIV” in the equation IN1 DIV IN2=Q.	INT, DINT, UINT, REAL, LREAL	All except S, SA, SB, SC	No
Q	The quotient of IN1/IN2. If an overflow occurs, the result is the largest value with the proper sign.	INT, DINT, UINT, REAL or LREAL variable	All except S, SA, SB, SC and constant	No

Modulus



Divides input IN1 by input IN2 and outputs the remainder of the division to Q.

All three operands must be of the same data type. The sign of the result is always the same as the sign of input parameter IN1. Output Q is calculated using the formula:

Q = IN1-((IN1 DIV IN2) * IN2)

where DIV produces an integer number.

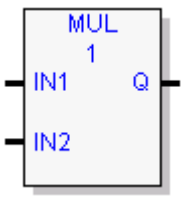
The result is output to Q unless one of the following invalid conditions occurs:

- 0 divided by 0 (Results in an application fault.)
- IN1 and/or IN2 is NaN (Not a Number)

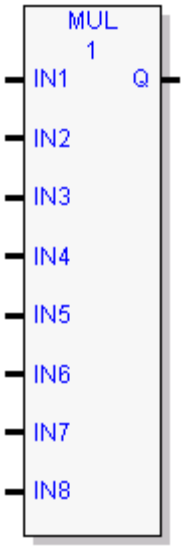
Operands for Modulus Function

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1	The value to be divided to obtain the remainder; the value to the left of “MOD” in the equation IN1 MOD IN2=Q.	INT, DINT, UINT	All except S, SA, SB, SC	No
IN2	The value to divide IN1 with; the value to the right of “MOD” in the equation IN1 MOD IN2=Q.	INT, DINT, UINT	All except S, SA, SB, SC	No
Q	The remainder of IN1/IN2.	INT, DINT, UINT variable	All except S, SA, SB, SC and constant	No

Multiply



Minimum number of inputs = 2



Maximum number of inputs = 8.

Multiplies two through eight operands (IN1 … IN8) of the same data type and stores the result in the output variable assigned to Q, also of the same data type.

The output is calculated when the function is performed without overflow, unless an invalid operation occurs.

If an overflow occurs, the result wraps around.

Mnemonic	*Operation*	*Displays as*
INT	Q(16 bit) = IN1(16 bit) * IN2(16 bit)	base 10 number with sign, up to 5 digits long
DINT	Q(32 bit) = IN1(32 bit) * IN2(32 bit)	base 10 number with sign, up to 10 digits long
REAL	Q(32 bit) = IN1(32 bit) * IN2(32 bit)	base 10 number, sign and decimals, up to 8 digits long (excluding the decimals)
UINT	Q(16 bit) = IN1(16 bit) * IN2(16 bit)	base 10 number, unsigned, up to 5 digits long

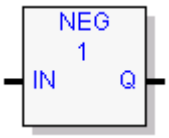
Operands for Multiply

Parameter	*Description*	*Allowed Types*	*Allowed Operands*	*Optional*
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1 … IN8	The values to multiply. Must be the same data type as Q.	INT, DINT, UINT, REAL	All except S, SA, SB, SC	No
Q	The result of the multiplication.	INT, DINT, UINT, REAL variable	All except S, SA, SB, SC and constant	No

Properties for Multiply

Property	*Valid Range*
Number of Inputs	2 to 8

Negate

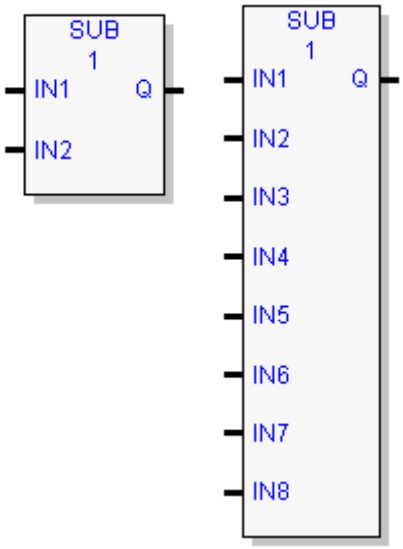


Multiplies a number by –1 and places the result in the output location, Q.

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN	The value to be negated.	INT, DINT, REAL	All except S, SA, SB, SC	No
Q	The result, -1(IN)	INT, DINT, REAL variable	All except S, SA, SB, SC and constant	No

Subtract



Minimum number of inputs = 2

Maximum number of inputs = 8.

Subtracts the operands IN2 …IN8 from the operand IN1 and stores the result in the output variable assigned to Q.

The calculation is carried out when SUB is performed without overflow, unless an invalid operation occurs.

If a SUB operation results in overflow, the result wraps around. For example:

- If a SUB_DINT, SUB_INT or SUB_REAL operation is performed on 32768 - 1, Q will be set to -32767.

If a SUB_UINT operation results in a negative number, Q wraps around. (For example, a result of –1 sets Q to 65535.)

Mnemonic	*Operation*	*Displays as*
SUB_INT	Q(16 bit) = IN1(16 bit) – IN2(16 bit)	base 10 number with sign, up to 5 digits long
SUB_DINT	Q(32 bit) = IN1(32 bit) – IN2(32 bit)	base 10 number with sign, up to 10 digits long
SUB_REAL	Q(32 bit) = IN1(32 bit) – IN2(32 bit)	base 10 number, sign and decimals, up to 8 digits long (excluding the decimals)
SUB_UINT	Q(16 bit) = IN1(16 bit) – IN2(16 bit)	base 10 number, unsigned, up to 5 digits long

Operands for Subtract

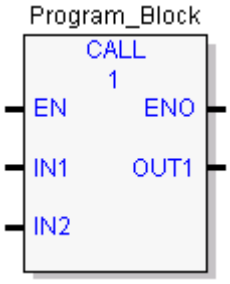
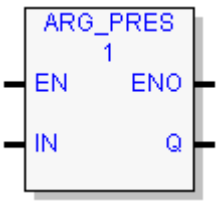
Parameter	*Description*	*Allowed Types*	*Allowed Operands*	*Optional*
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN1	The value to subtract from.	DINT, INT, REAL, UINT	All except S, SA, SB, SC	No
IN2 … IN8	The value(s) to subtract from IN1. Must be the same data type as IN1.		All except S, SA, SB, SC	No
Q	The result of the subtraction. Must be the same data type as IN1.	DINT, INT, REAL, UINT variable	All except S, SA, SB, SC and constant	No

Properties for Subtract

Property	*Valid Range*
Number of Inputs	2 to 8

Program Flow Functions

The program flow functions limit program execution or change the way the CPU executes the application program.

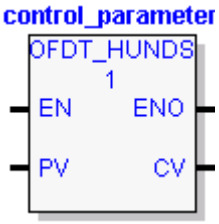
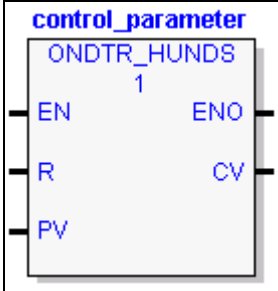
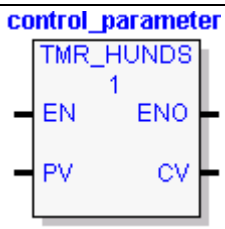
Function		Description
 Non-parameterized CALL	 Parameterized CALL. May call a parameterized external block or a parameterized block.	The CALL function causes the logic execution to go immediately to the designated program block, external C block (parameterized or not), or parameterized block and execute it. After the block's execution is complete, control returns to the point in the logic immediately following the CALL instruction. For details, see "Program Flow Functions" in chapter 8.
		The ARG_PRES (Argument Present) function determines whether a parameter value was present when the function block instance of the parameter was invoked. For details, see "Program Flow Functions" in chapter 8.

Timers

This section describes the PACSystems timing functions that are implemented in the FBD language.

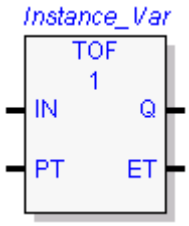
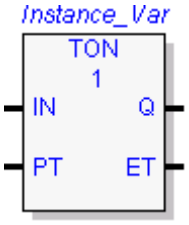
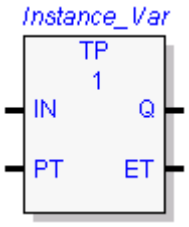
Built-in Timer Function Blocks

These function blocks use WORD Array instance data. The parameter that appears above the function block is a one-dimensional, three-word array in %R, %W, %P, %L, or symbolic memory that the timer uses to store its current value, preset value and control word.

Function	Description
	Off Delay Timer. The timer's Current Value (CV) resets to zero when its enable parameter (EN) is set to ON.. CV increments while EN is OFF. When CV=PV (Preset Value), ENO is set to OFF until EN is set to ON again. Other OFDT functions: OFDT_SEC OFDT_TENTHS OFDT_THOUS For details, see “Timers” in chapter 8.
	On Delay Stopwatch Timer. Retentive on delay timer. Increments while EN is ON and holds its value when EN is OFF. ONDTR_SEC ONDTR_TENTHS ONDTR_THOUS For details, see “Timers” in chapter 8.
	On Delay Timer. Simple on delay timer. Increments while EN is ON and resets to zero when EN is OFF. TMR_SEC TMR_TENTHS TMR_THOUS For details, see “Timers” in chapter 8.

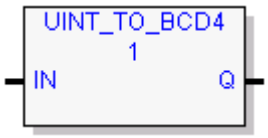
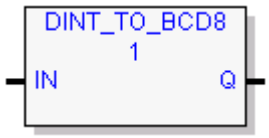
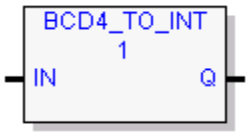
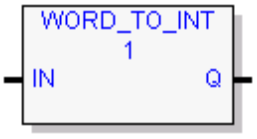
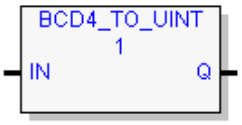
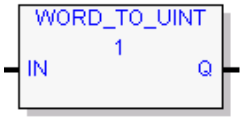
Standard Timer Function Blocks

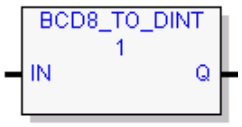
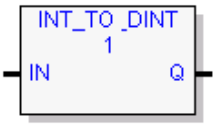
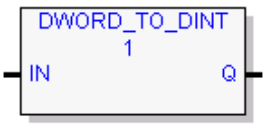
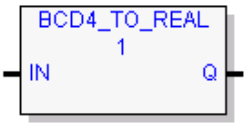
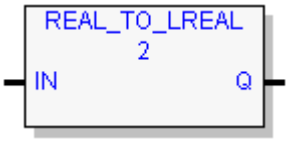
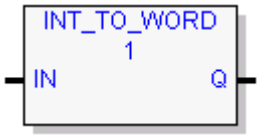
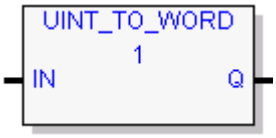
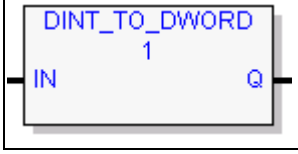
These functions blocks use Structure Variable instance data. Each invocation of a timer has associated instance data that persists from one execution of the timer to the next. Instance variables are automatically located in symbolic memory. (You cannot specify an address.) You can specify a stored value for each element. The user logic cannot modify the values.

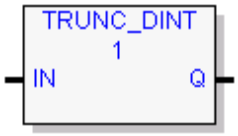
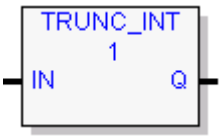
Function	Description
	Timer Off Delay. When the input IN transitions from ON to OFF, the timer starts timing until a specified period of time has elapsed, then sets the output Q to OFF. For details, see “Timers” in chapter 8.
	Timer On Delay. When the input IN transitions from OFF to ON, the timer starts timing until a specified period of time has elapsed, then sets the output Q to ON. For details, see “Timers” in chapter 8.
	Timer Pulse. When the input IN transitions from OFF to ON, the timer sets the output Q to ON for a specified time interval. For details, see “Timers” in chapter 8.

Type Conversion Functions

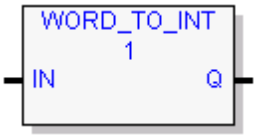
The Conversion functions change a data item from one number format (data type) to another. Many programming instructions, such as math functions, must be used with data of one type. As a result, data conversion is often required before using those instructions.

Function	Description
Convert Angles	
	DEG_TO_RAD: Converts degrees to radians. RAD_TO_DEG: Converts radians to degrees. For details, see “Conversion Functions” in chapter 8.
Convert to BCD4 (4-digit Binary-Coded-Decimal)	
	UINT_TO_BDC4: Converts UINT (16-bit unsigned integer) to BCD4. INT_TO_BCD4: Converts INT (16-bit signed integer) to BCD4. For details, see “Conversion Functions” in chapter 8.
Convert to BCD8 (8-digit Binary-Coded-Decimal)	
	DINT_TO_BD8: Converts DINT (32-bit signed integer) to BCD8. For details, see “Conversion Functions” in chapter 8.
Convert to INT (16-bit signed integer)	
	BCD4_TO_INT: Converts BCD to INT. UINT_TO_INT: Converts UINT to INT DINT_TO_INT: Converts DINT to INT.. REAL_TO_INT: Converts REAL to INT. For details, see “Conversion Functions” in chapter 8.
	Converts a 16-bit string (WORD) value to INT. For details, see page 9-34.
Convert to UINT (16-bit unsigned integer)	
	BCD4_TO_UINT: Converts BCD4 to UINT. INT_TO_UINT: Converts INT to UINT. DINT_TO_UINT: Converts DINT to UINT. REAL_TO_UINT: Converts REAL to UINT. For details, see “Conversion Functions” in chapter 8.
	WORD_TO_UINT: Converts a 16-bit string (WORD) value to UINT. For details, see page 9-35.

Function	Description
Convert to DINT (32-bit signed integer)	
	BCD8_TO_DINT: Converts BCD8 to DINT. UINT_TO_DINT: Converts UINT to DINT. For details, see “Conversion Functions” in chapter 8.
	INT_TO_DINT: Converts INT to DINT. REAL_TO_DINT: Converts REAL (32-bit signed real or floating-point values) to DINT. For details, see “Conversion Functions” in chapter 8.
	DWORD_TO_DINT: Converts a 32-bit bit string (DWORD) value to DINT. For details, see page 9-35.
Convert to REAL (32-bit signed real or floating-point values)	
	BCD4_TO_REAL: Converts BCD4 to REAL. BCD8_TO_REAL: Converts BCD8 to REAL. UINT_TO_REAL: Converts UINT to REAL. INT_TO_REAL: Converts INT to REAL. DINT_TO_REAL: Converts DINT to REAL. LREAL_TO_REAL: Converts LREAL to REAL. For details, see “Conversion Functions” in chapter 8.
Convert to LREAL(64-bit signed real or floating-point values)	
	Converts a REAL value to LREAL. For details, see “Conversion Functions” in chapter 8.
Convert to WORD (16-bit string)	
	Converts an INT (16-bit signed integer) value to a WORD value. For details, see page 9-36.
	Converts an unsigned single-precision integer (UINT) to WORD. For details, see page 9-36.
Convert to DWORD (32-bit bit string)	
	Converts a double-precision signed integer (DINT) value to DWORD. For details, see page 9-36.

Function	Description
Truncate	
	Rounds a REAL (32-bit signed real or floating-point) number down to a DINT number For details, see “Conversion Functions” in chapter 8.
	Rounds a REAL (32-bit signed real or floating-point) number down to an INT number For details, see “Conversion Functions” in chapter 8.

Convert WORD to INT



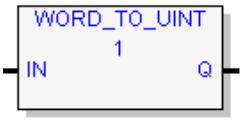
Converts the input data into the equivalent single-precision signed integer (INT) value, which it outputs to Q. This function does not change the original input data. The output data can be used directly as input for another program function, as in the examples.

The function passes data to Q, unless the data is out of range (0 through +65,535).

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN	The value to convert to INT.	WORD	All except S, SA, SB, and SC	No
Q	The INT equivalent value of the original value in IN.	INT	All except S, SA, SB, SC and constant	No

Convert WORD to UINT



These functions convert the input data into the equivalent single-precision unsigned integer (UINT) value, which it outputs to Q.

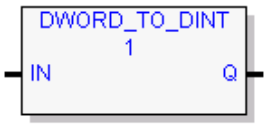
The conversion to UINT does not change the original data. The output data can be used directly as input for another program function, as in the example.

The function passes the converted data to Q, unless the resulting data is outside the range 0 to +65,535.

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN	The value to convert to UINT.	WORD	All except S, SA, SB, and SC	No
Q	The UINT equivalent value of the original input value in IN.	UINT	All except S, SA, SB, SC and constant	No

Convert DWORD to DINT



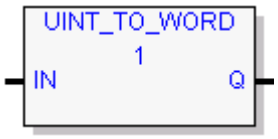
Converts DWORD data into the equivalent signed double-precision integer (DINT) value and stores the result in Q. The conversion to DINT does not change the original data.

The output data can be used directly as input for another program function. The function passes data to Q unless the data is out of range.

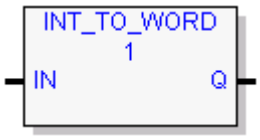
Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN	The value to convert to DINT.	DWORD	All except S, SA, SB, and SC	No
Q	The DINT equivalent value of the original input value in IN.	UINT	All except S, SA, SB, SC and constant	No

Convert INT or UINT to WORD



Converts an unsigned single-precision integer (UINT) operand IN to a 16-bit bit string (WORD) value and stores the result in the variable assigned to Q.



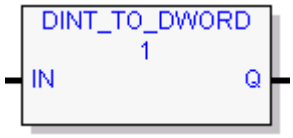
Converts a 16-bit signed integer (INT) operand IN to a 16-bit bit string (WORD) value and stores the result in the variable assigned to Q.

The output data can be used directly as input for another program function. The function passes data to Q unless the data is out of range.

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN	The value to convert to WORD.	INT or UINT, depending on function	All except S, SA, SB, and SC	No
Q	The WORD equivalent value of the original value in IN. 0 ≤ Q ≤ 65,535.	WORD	All except S, SA, SB, SC and constant	No

Convert DINT to DWORD



When DINT_TO_DWORD receives data, it converts the input double-precision signed integer (DINT) data into the equivalent DWORD (32-bit bit string) value, which it outputs to Q. DINT_TO_DWORD does not change the original DINT data.

The output data can be used directly as input for another program function. The function passes data to Q unless the data is out of range.

Operands

Parameter	Description	Allowed Types	Allowed Operands	Optional
Solve Order	Calculated by the FBD editor.	NA	NA	No
IN	The value to convert to DWORD.	DINT	All except S, SA, SB, and SC	No
Q	The DWORD equivalent value of the original value in IN. 0 ≤ Q ≤ 4,294,967,295.	DWORD	All except S, SA, SB, SC and constant	No

Chapter 10 Service Request Function

Use a Service Request function to request one of the following control system services:

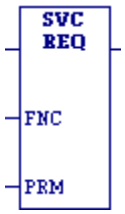
Service Request	*Description*	*Page*
SVC_REQ 1	Change/Read Constant Sweep Timer	10-4
SVC_REQ 2	Read Window Modes and Times Values	10-6
SVC_REQ 3	Change Controller Communications Window Mode and Timer Value	10-7
SVC_REQ 4	Change Backplane Communications Window Mode and Timer Value	10-8
SVC_REQ 5	Change Background Task Window Mode and Timer Value	10-9
SVC_REQ 6	Change/Read Number of Words to Checksum	10-10
SVC_REQ 7	Read or Change the Time-of-Day Clock	10-12
SVC_REQ 8	Reset Watchdog Timer	10-19
SVC_REQ 9	Read Sweep Time from Beginning of Sweep - milliseconds	10-19
SVC_REQ 10	Read Target Name	10-20
SVC_REQ 11	Read Controller ID	10-21
SVC_REQ 12	Read PLC Run State	10-22
SVC_REQ 13	Shut Down (Stop) PLC	10-23
SVC_REQ 14	Clear PLC or I/O Fault Tables	10-24
SVC_REQ 15	Read Last-Logged Fault Table Entry	10-25
SVC_REQ 16	Read Elapsed Time Clock - microseconds	10-27
SVC_REQ 17	Mask/Unmask I/O Interrupt	10-29
SVC_REQ 18	Read I/O Override Status	10-31
SVC_REQ 19	Set Run Enable/Disable	10-32
SVC_REQ 20	Read Fault Tables	10-33
SVC_REQ 21	User-Defined Fault Logging	10-38
SVC_REQ 22	Mask/Unmask Timed Interrupts	10-40
SVC_REQ 23	Read Master Checksum	10-41
SVC_REQ 24	Reset Module	10-42
SVC_REQ 25	Disable/Enable EXE Block and Standalone C Program Checksums	10-43
SVC_REQ 26	Role Switch (Redundancy)	*
SVC_REQ 27	Write to Reverse Transfer Area (Hot Standby Redundancy)	*
SVC_REQ 28	Read from Reverse Transfer Area (Hot Standby Redundancy)	*
SVC_REQ 29	Read Elapsed Power Down Time	10-44
SVC_REQ 32	Suspend/Resume I/O Interrupt	10-45
SVC_REQ 43	Disable Data Transfer Copy in Backup Unit (Hot Standby Redundancy)	*
SVC_REQ 45	Skip Next I/O Scan	10-47
SVC_REQ 50	Read Elapsed Time Clock – nanoseconds	10-48
SVC_REQ 51	Read Sweep Time from Beginning of Sweep - nanoseconds	10-50
SVC_REQ 55	Set Application Redundancy Mode (non-Hot Standby Redundancy)	*

*For information on Service Requests used in CPU HSB redundancy applications, refer to the *PACSystems Hot Standby CPU Redundancy User's Guide,* GFK-2308. For non-HSB applications, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224.

Operation of SVC_REQ Function

PACSystems supports the Service Request function in LD and FBD.

Ladder Diagram



When SVC_REQ receives power flow, it requests the CPU to perform the special service identified by the FNC operand.

Parameters for SVC_REQ are located in the parameter block, which begins at the reference identified by the PRM operand. The number of 16-bit references required depends on the type of special PLC service being requested. The parameter block is used to store both the function's inputs and outputs.

SVC_REQ passes power flow unless an incorrect function number, incorrect parameters, or out-of-range references are specified. Specific SVC_REQ functions may have additional causes for failure.

Because the service request continues to be invoked each time power flow is enabled to the function, additional enable/disable logic preceding the request may be necessary, depending upon the application. (For example, repeated calling of SVC_REQ 24 would continually reset a module, probably not the intended behavior.) In many cases a transition contact or coil will be sufficient. Alternatively, you could use more complex logic, such as having the function contained within a block that is only called a single time.

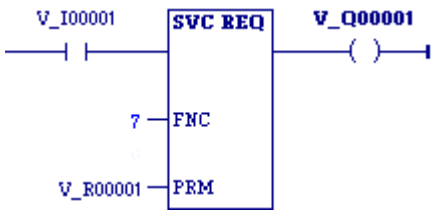
Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

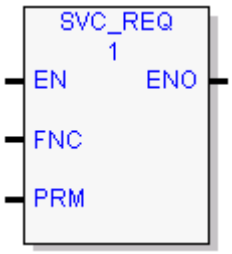
Operand	*Data Type*	*Memory Area*	*Description*
FNC	INT variable or constant	All except %S - %SC	Function number; Service Request number. The constant or reference that identifies the requested service.
PRM	WORD variable	All except flow, %S - %SC and constant	The first WORD in the parameter block for the requested service. Successive 16-bit locations store additional parameters.

Example

When the enabling input %I0001 is ON, SVC_REQ function number 7 is called, with the parameter block starting at %R0001. If the operation succeeds, output coil %Q0001 is set ON.



Function Block Diagram



The SVC_REQ function requests the CPU to perform the special service identified by the FNC operand.

Parameters for SVC_REQ are located in the parameter block, which begins at the reference identified by the PRM operand. The number of 16-bit references required depends on the type of special PLC service being requested. The parameter block is used to store both the function's inputs and outputs.

Operands

Note: Indirect referencing is available for all register references (%R, %P, %L, %W, %AI, and %AQ).

Parameter	*Description*	*Allowed Types*	*Allowed Operands*	*Optional*
Solve Order	Calculated by the FBD editor.	NA	NA	No
EN	Enable input. When set to ON, the SVC_REQ executes	BOOL	data flow, I, Q, M, T, G, S, SA, SB, SC, discrete symbolic, I/O variable	No
		Bit reference in a non-BOOL variable	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	
FNC	Function number; Service Request number. The constant or variable that identifies the requested service.	INT, DINT, UINT, WORD, DWORD	All except %S - %SC You can use data flow only if the parameter block requires only one WORD If you use a symbolic variable or an I/O variable, ensure that its Array Dimension 1 property is set to a value large enough to contain the entire parameter block.	No
PRM	The first word in the parameter block for the requested service. Successive 16-bit locations store additional parameters.	INT, DINT, UINT, WORD, DWORD	All except flow, %S - %SC and constant	No
ENO	Set to ON unless an incorrect function number, incorrect parameters, or out-of-range references are specified. Specific SVC_REQ functions may have additional causes for failure.	BOOL	data flow, I, Q, M, T, G, non-discrete symbolic, I/O variable	Yes
		Bit reference in a non-BOOL variable.	I, Q, M, T, G, R, P, L, AI, AQ, W, non-discrete symbolic, I/O variable	

SVC_REQ 1: Change/Read Constant Sweep Timer

Use SVC_REQ function 1 to:

- Disable Constant Sweep mode
- Enable Constant Sweep mode and use the old Constant Sweep timer value
- Enable Constant Sweep mode and use a new Constant Sweep timer value
- Set a new Constant Sweep timer value only
- Read Constant Sweep mode state and timer value.

The parameter block has a length of two words used for both input and output.

SVC_REQ executes successfully unless:

- A number other than 0, 1, 2, or 3 is entered as the requested operation:
- The scan time value is greater than 2550 ms (2.55 seconds)
- Constant sweep time is enabled with no timer value programmed or with an old value of 0 for the timer.

To disable Constant Sweep mode:

Enter SVC_REQ 1 with this parameter block:

Address	0
Address + 1	Ignored

To enable Constant Sweep mode and use the old timer value:

Enter SVC_REQ 1 with this parameter block:

Address	1
Address + 1	0

If the timer value does not already exist, entering 0 causes the function to set the OK output to OFF.

To enable Constant Sweep mode and use a new timer value:

Enter SVC_REQ 1 with this parameter block:

Address	1
Address + 1	New timer value **Note:** If the timer value does not already exist, entering 0 causes the function to set the OK output to OFF.

To change the timer value without changing the selection for sweep mode state:

Enter SVC_REQ 1 with this parameter block:

Address	2
Address + 1	New timer value

To read the current timer state and value without changing either:

Enter SVC_REQ 1 with this parameter block:

Address	3
Address + 1	ignored

Output

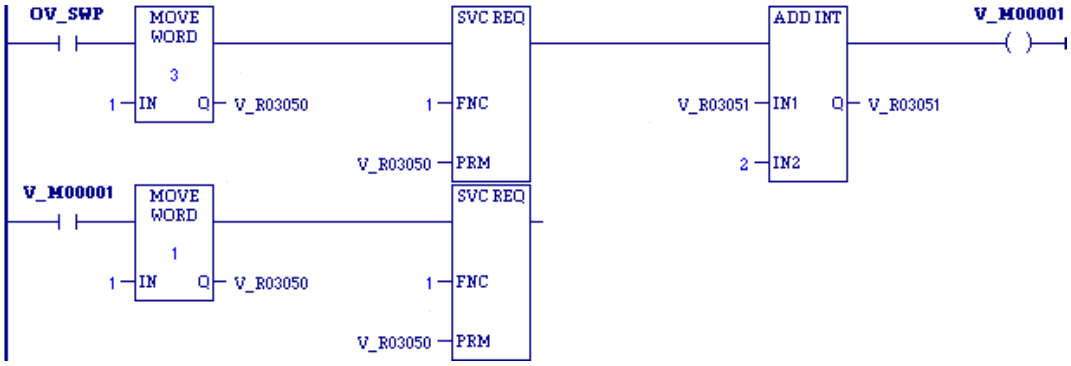
SVC_REQ 1 returns the timer state and value in the same parameter block references:

Address	0 = Normal Sweep 1 = Constant Sweep
Address + 1	Current timer value

If the word address + 1 contains the hexadecimal value FFFF, no timer value has been programmed.

Example

If contact OV_SWP is set, the Constant Sweep Timer is read, the timer is increased by two milliseconds, and the new timer value is sent back to the CPU. The parameter block is at location %R3050. The example logic uses discrete internal coil %M0001 as a temporary location to hold the successful result of the first rung line. On any sweep in which OV_SWP is not set, %M00001 is turned off.



SVC_REQ 2: Read Window Modes and Times Values

Use SVC_REQ 2 to obtain the current window mode and time values for the controller communications window and the backplane communications and the background task window.

The parameter block has a length of three words. All parameters are output parameters. It is not necessary to enter values in the parameter block to program this function.

Output

Address	Window	High Byte	Low Byte
address	Controller Communications Window	Mode	Value in ms
address+1	Backplane Communications Window	Mode	Value in ms
address+2	Background Window	Mode	Value in ms

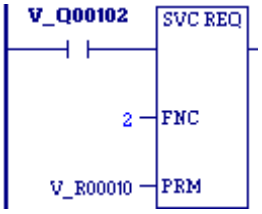
Note: A window is disabled when the time value is zero.

Mode Values

Mode Name	Value	Description
Limited Mode	0	The execution time of the window is limited to its respective default value or to a value defined using SVC_REQ 3 for the controller communications window or SVC_REQ 4 for the systems communications window. The window will terminate when it has no more tasks to complete.
Constant Mode	1	Each window will operate in a Run to Completion mode, and the CPU will alternate among the three windows for a time equal to the sum of each window's respective time value. If one window is placed in Constant mode, the remaining two windows are automatically placed in Constant mode. If the CPU is operating in Constant Window mode and a particular window's execution time is not defined using the associated SVC_REQ function, the default time for that window is used in the constant window time calculation.
Run to Completion Mode	2	Regardless of the window time associated with a particular window, whether default or defined using a service request function, the window will run until all tasks within that window are completed.

Example

When %Q00102 is set, the CPU places the current time values of the windows in the parameter block starting at location %R0010.



SVC_REQ 3: Change Controller Communications Window Mode

Use SVC_REQ 3 to change the controller communications window mode and timer value. The change takes place during the next CPU sweep after the function is called.

The parameter block has a length of one word.

SVC_REQ 3 executes unless a mode other than 0 (Limited) or 2 (Run to Completion) is selected.

To disable the controller communications window:

Enter SVC_REQ 3 with this parameter block:

Address	***High Byte***	***Low Byte***
Address	0	0

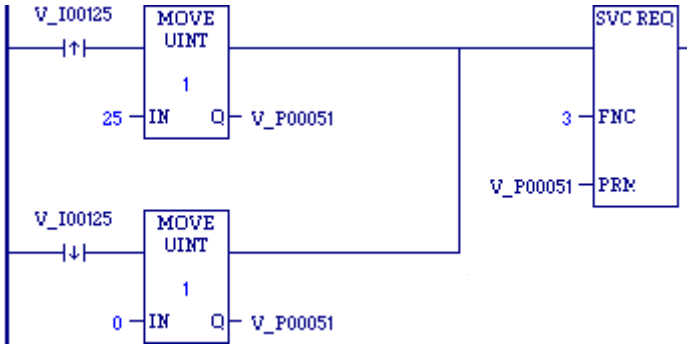
To re-enable or change the controller communications window mode:

Enter SVC_REQ 3 with this parameter block:

Address	***High Byte***	***Low Byte***
Address	Mode: 0 = Limited 2 = Run to Completion	1ms ≤ value ≤ 255ms in 1ms increments

Example

When enabling input %I00125 transitions on, the controller communications window is enabled and assigned a value of 25 ms. When the contact transitions off, the window is disabled. The parameter block is in global memory location %P00051.



SVC_REQ 4: Change Backplane Communications Window Mode and Timer Value

Use SVC_REQ 4 to change the Backplane Communications window mode and timer value. The change takes place during the next CPU sweep after the function is called.

SVC_REQ 4 executes unless a mode other than 0 (Limited) or 2 (Run to Completion) is selected.

The parameter block has a length of one word.

To disable the Backplane Communications window:

Enter SVC_REQ 4 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	0	0

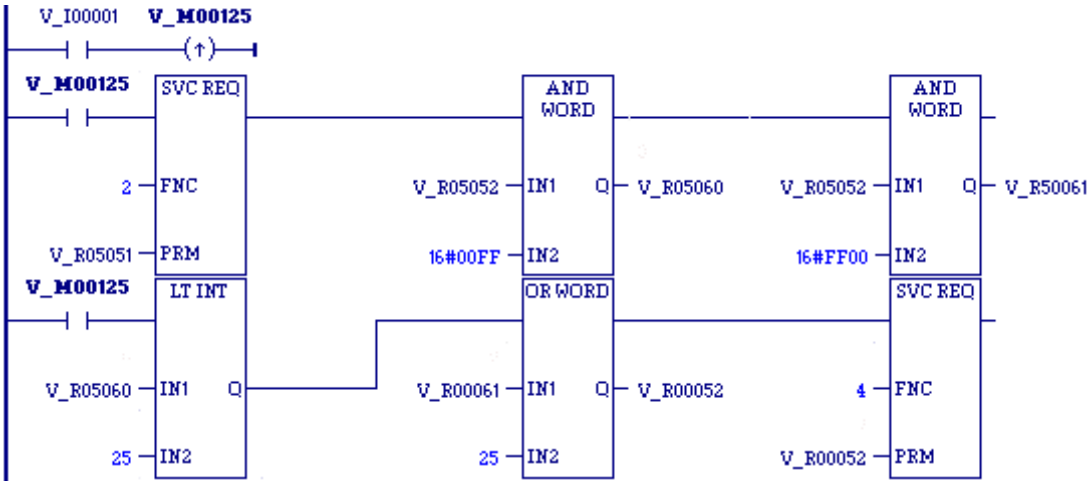
To enable the Backplane Communications window mode:

Enter SVC_REQ 4 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode 0 = Limited 2 = Run to Completion	1ms ≤ value ≤ 255ms

Example

When enabling output %M0125 transitions on, the mode and timer value of the Backplane Communications window is read. If the timer value is greater than or equal to 25 ms, the value is not changed. If it is less than 25 ms, the value is changed to 25 ms. In either case, when the rung completes execution the window is enabled. The parameter block for all three windows is at location %R5051. Since the mode and timer for the Backplane Communications window is the second value in the parameter block returned from the Read Window Values function (SVC_REQ 2), the location of the existing window time for the Backplane Communications window is in the low byte of %R5052.



SVC_REQ 5: Change Background Task Window Mode and Timer Value

Use SVC_REQ 5 to change the Background Task window mode and timer value. The change takes place during the next CPU sweep after the function is called.

SVC_REQ 5 executes unless a mode other than 0 (Limited) or 2 (Run-to-Completion) is selected.

The parameter block has a length of one word.

To disable the Background Task window:

Enter SVC_REQ 5 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	0	0

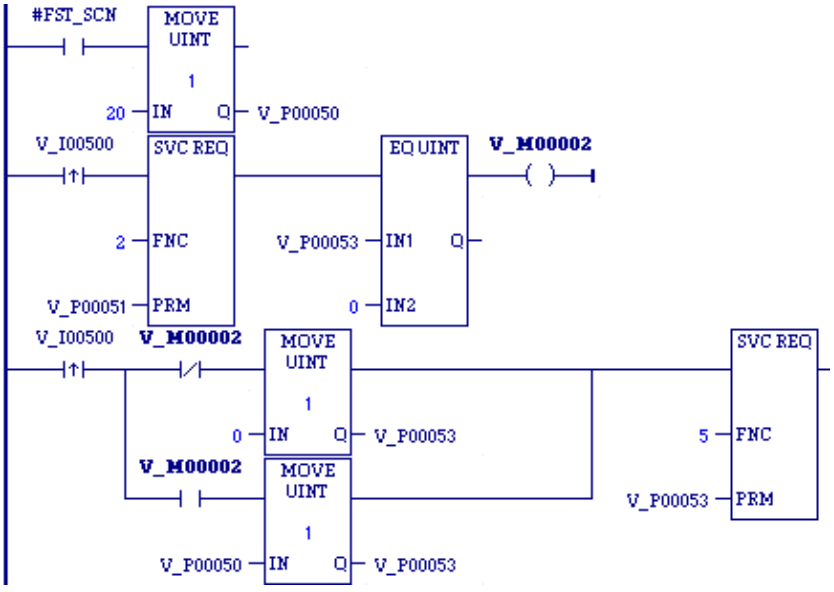
To enable the Background Task window mode:

Enter SVC_REQ 5 with this parameter block:

Address	*High Byte*	*Low Byte*
Address	Mode 0 = Limited 2 = Run to Completion	1ms ≤ value ≤ 255ms

Example

When enabling contact #FST_SCN is set in the first scan, the MOVE function establishes a value of 20ms for the Background task window, using a parameter block beginning at %P00050. Later in the program, when input %I00500 transitions on, the state of the Background task window toggles on and off. The parameter block for all three windows is at location %P00051. The time for the Background task window is the third value in the parameter block returned from the Read Window Values function (function #2); therefore, the location of the existing window time for the Background window is %P00053.



SVC_REQ 6: Change/Read Number of Words to Checksum

Use SVC_REQ 6 to read the current word count in the program to be checksummed or set a new word count. By default, 16 words are checked. The function is successful unless some number other than 0 or 1 is entered as the requested operation.

The parameter block has a length of 2 words.

To read the word count:

Enter a zero in the first word of the parameter block.

Address	0
Address + 1	Ignored

The function returns the current checksum (word count) in the second word of the parameter block. No range is specified for the read function; the value returned is the number of words currently being checksummed.

Address	0
Address + 1	Current word count

To set a new word count:

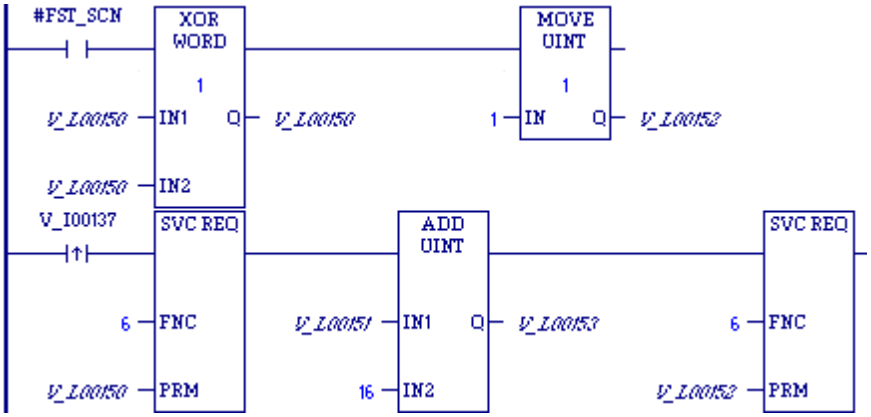
Enter a one in the first word of the parameter block and the new word count in the second word.

Address	1
Address + 1	New word count

The CPU changes the number of words to be checksummed to the value given in the second word of the parameter block, rounded up to the next multiple of 8. To disable checksumming, set the new word count to 0.

Example

When enabling contact #FST_SCN is set, the parameter blocks for the checksum task function are built. Later in the program, when input %I00137 transitions on, the number of words being checksummed is read from the CPU operating system. This number is increased by 16, with the results of the ADD_UINT function being placed in the “hold new count for set” parameter. The second service request block requests the CPU to set the new word count.



The example parameter blocks are located at address %L00150. They have the following contents:

%L00150	0 = read current count
%L00151	hold current count
%L00152	1 = set current count
%L00153	hold new count for set

SVC_REQ 7: Read or Change the Time-of-Day Clock

Use SVC_REQ 7 to read or change the time of day clock in the CPU. The function is successful unless:

- An invalid number is entered for the requested operation.
- An invalid data format is specified.
- Data is provided in an unexpected format.

Parameter Block Formats

In the first two words of the parameter block, you specify whether to read or set the time and date, and which format to use.

Address	*2-Digit Year Format*	*4-Digit Year Format*
Address (word 1)	0 = read time and date	0 = read time and date
	1 = set time and date	1 = set time and date
Address+1 (word 2)	0 = numeric data format	80h – numeric data format
	1 = BCD format	81h = BCD format
	2 = unpacked BCD format	82h = unpacked BCD format
	3 = packed ASCII format (with embedded spaces and colons)	83h = packed ASCII format
	4 = POSIX format	n/a
Address+2 (word 3) to the end	Data	Data

Words 3 to the end of the parameter block contain output data returned by a read function, or new data being supplied by a change function. In both cases, format of these data words is the same. When reading the date and time, words (address + 2) to the end of the parameter block are ignored on input.

The format and length of the parameter block depends on the data format and number of digits required for the year:

Data Format and N-digit Year	*Length of parameter block (number of words)*
BCD, 2-digit year	6
BCD, 4-digit year	6
POSIX format	6
Unpacked BCD 2	9
Unpacked BCD 4	10
Numeric (2 and 4 digit years)	9
Packed ASCII, 2-digit year	12
Packed ASCII, 4-digit year	13

In any format:

- Hours are stored in 24-hour format.
- Day of the week is a numeric value ranging from 1 (Sunday) to 7 (Saturday).

Value	*Day of the Week*
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

BCD, 2-Digit Year

In BCD format, each time and date item occupies one byte, so the parameter block has six words. The last byte of the sixth word is not used. When setting the date and time, this byte is ignored; when reading date and time, the function returns a null character (00).

Parameter Block Format	*Address*	*Example (Sun., July 3, 2005, at 2:45:30 p.m. = 14:45:30 in 24-hour format)*
1 = change or 0 = read	address	0 (read)
1 (BCD format)	address+1	1 (BCD format)

High Byte	*Low Byte*	*Address*	*High Byte*	*Low Byte*
month	year	address+2	07 (July)	05 (year)
hours	day of month	address+3	14 (hours)	03 (day)
seconds	minutes	address+4	30 (seconds)	45 (minutes)
(null)	day of week	address+5	00	01 (Sunday)

BCD, 4-Digit Year

In this format, all bytes are used.

Parameter Block Format	*Address*	*Example (Sun., July 3, 2005, at 2:45:30 p.m. = 14:45:30 in 24-hour format)*
1 = change or 0 = read	address	00 (read)
81h (BCD format, 4-digit)	address+1	81h (BCD format, 4-digit)

High Byte	*Low Byte*	*Address*	*High Byte*	*Low Byte*
year	year	address+2	20 (year)	05 (year)
day of month	month	address+3	03 (day)	07 (July)
minutes	hours	address+4	45 (minutes)	14 (hours)
day of week	seconds	address+5	01 (Sunday)	30 (seconds)

POSIX

The POSIX format of the Time-of-Day clock uses two signed 32-bit integers (two DINTs) to represent the number of seconds and nanoseconds since midnight January 1, 1970. Reading the clock in POSIX format might make it easier for your application to calculate time differences. This format can also be useful if your application communicates to other devices using the POSIX time format. To read and/or change the date and time using POSIX format, enter SVC_REQ 7 with this parameter block:

Parameter Block Format	***Address***	***Example December 1, 2000 at 12 noon***
1 = change or 0 = read	address	0
4 (POSIX format)	address+1	4
Seconds (LSW)	address+2	975,672,000
(MSW)	address+3	
Nanoseconds (LSW)	address+4	0
(MSW)	address+5	

The PACSystems CPU's maximum POSIX clock value is F48656FE (hexadecimal) seconds and 999,999,999 (decimal) nanoseconds, which corresponds to December 31st, 2099 at 11:59 pm. This is the maximum POSIX value that SVC_REQ 7 will accept for changing the clock. This is also the maximum POSIX value SVC_REQ 7 will return once the Time-Of-Day clock passes this date.

If SVC_REQ 7 receives an invalid POSIX time to write to the clock, it does not change the Time-Of-Day clock and disables its power-flow output.

Note: When reading the PACSystems CPU clock in POSIX format, the data returned is not easily interpreted by a human viewer. If desired, it is up to the application logic to convert the POSIX time into year, month, day of month, hour, and seconds.

Unpacked BCD (2-Digit Year)

In Unpacked BCD format, each digit of the time and date items occupies the low-order four bits of a byte. The upper four bits of each byte are always zero. This format requires nine words. Values are hexadecimal.

Parameter Block Format	***Address***	***Example (Thurs., Dec. 8, 2002, at 9:34:57 a.m.)***
1 = change or 0 = read	address	0h
2 (Unpacked BCD format)	address+1	2h

High Byte	***Low Byte***		***High Byte***	***Low Byte***
	year	address+2	00h	02h
	month	address+3	01h	02h
	day of month	address+4	02h	08h
	hours	address+5	00h	09h
	minutes	address+6	03h	04h
	seconds	address+7	05h	07h
	day of week	address+8	00h	05h

Unpacked BCD (4-Digit Year)

In Unpacked BCD format, each digit of the time and date items occupies the low-order four bits of a byte. The upper four bits of each byte are always zero. This format requires nine words. Values are hexadecimal.

Parameter Block Format	Address	Example (Thurs., Dec. 8, 2002, at 9:34:57 a.m.)
1 = change or 0 = read	address	0h
82h (Unpacked 4-digit BCD format)	address+1	82h

High Byte	Low Byte		High Byte	Low Byte
	year	address+2	00h	02h
	month	address+3	01h	02h
	day of month	address+4	00h	08h
	hours	address+5	00h	09h
	minutes	address+6	03h	04h
	seconds	address+7	05h	07h
	day of week	address+8	00h	05h

Numeric, 2-Digit Year

In numeric format, the year, month, day of month, hours, minutes, seconds and day of week each occupy one unsigned integer. To read and/or change the date and time using the numeric format, enter SVCREQ function #7 with this parameter block:

Parameter Block Format	Address	Example Wed., June 15, 2005, at 12:15:30 a.m.
1 = change or 0 = read	address	0
0 (Numeric format, 2-digit year)	address+1	0

High Byte	Low Byte		Value
	year	address+2	05
	month	address+3	06
	day of month	address+4	15
	hours	address+5	12
	minutes	address+6	15
	seconds	address+7	30
	day of week	address+8	04

Numeric, 4-Digit Year

In numeric format, the year, month, day of month, hours, minutes, seconds and day of week each occupy one unsigned integer. To read and/or change the date and time using the numeric format, enter SVCREQ function #7 with this parameter block:

Parameter Block Format	Address	Example Wed., June 15, 2005, at 12:15:30 a.m.
1 = change or 0 = read	address	0
80h (Numeric format, 4 digit year)	address+1	80h

High Byte	Low Byte		Value
	year	address+2	2005
	month	address+3	06
	day of month	address+4	15
	hours	address+5	12
	minutes	address+6	15
	seconds	address+7	30
	day of week	address+8	04

Packed ASCII, 2-Digit Year

In Packed ASCII format, each digit of the time and date items is an ASCII formatted byte. Spaces and colons are embedded into the data to format it for printing or display. ASCII format for a 2-digit year requires 12 words in the parameter block. Values are hexadecimal.

Parameter Block Format	Address	Example (Mon., Oct. 5, 2005, at 11:13:25 p.m. = 23:13:25 in 24-hour format)
1 = change or 0 = read	address	0h (read)
3 (ASCII format)	address+1	3h (ASCII format)

High Byte	Low Byte		High Byte	Low Byte
year	year	address+2	35h (5)	30h (0)
month	(space)	address+3	31h (1)	20h (space)
(space)	month	address+4	20h (space)	30h (0)
day of month	day of month	address+5	35h (5)	30h (leading 0)
hours	(space)	address+6	32h (2)	20h (space)
: (colon)	hours	address+7	3Ah (:)	33h (3)
minutes	minutes	address+8	33h (3)	31h (1)
seconds	: (colon)	address+9	32h (2)	3Ah (:)
(space)	seconds	address+10	20h (space)	35h (5)
day of week	day of week	address+11	32h (2 = Mon.)	30h (leading 0)

Packed ASCII, 4-Digit Year

ASCII format for a 4-digit year requires 13 words in the parameter block. Values are hexadecimal.

Parameter Block Format	***Address***	***Example (Mon., Oct. 5, 2005, at 11:13:25 p.m. = 23:13:25 in 24-hour format)***
1 = change or 0 = read	address	0h (read)
83 (ASCII format)	address+1	83h (ASCII format, 4-digit)

High Byte	***Low Byte***		***High Byte***	***Low Byte***
year (hundreds)	year (thousands)	address+2	30h (0)	32h (2)
year (ones)	year (tens)	address+3	35h (5)	30h (0)
month (tens)	(space)	address+4	31h (1)	20h (space)
(space)	month (ones)	address+5	20h (space)	30h (0)
day of month (ones)	day of month (tens)	address+6	35h (5)	30h (leading 0)
hours (tens)	(space)	address+7	32h (2)	20h (space)
: (colon)	hours (ones)	address+8	3Ah (:)	33h (3)
minutes (ones)	minutes (tens)	address+9	33h (3)	31h (1)
seconds (tens)	: (colon)	address+10	32h (2)	3Ah (A)
(space)	seconds (ones)	address+11	20 (space)	35 (5)
day of week (ones)	day of week (tens)	address+12	32h (2 = Mon.)	30h (leading 0)

SVC_REQ 7

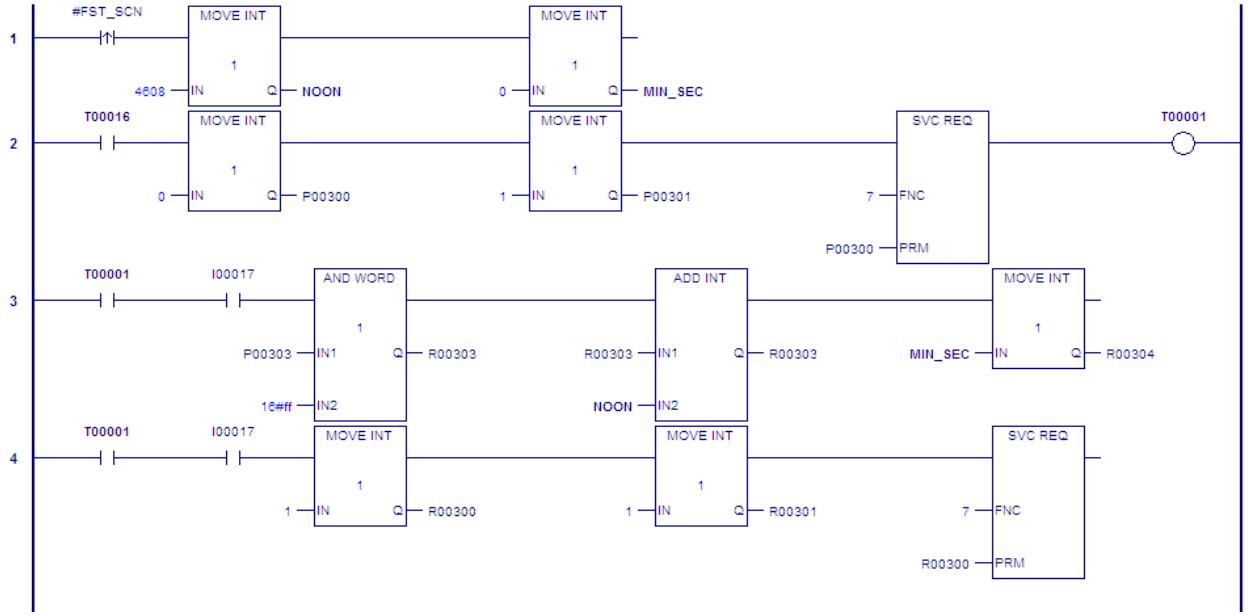
In this example, the time of day is set to 12:00 pm without changing the current year, BCD format requires six contiguous memory locations for the parameter block.

Rung 1 sets up the new time of day in two-digit year BCD format. It writes the value 4608 (equivalent to 12 00 BCD) to NOON and the value 0 to MIN_SEC.

Rung 2 requests the current date and time using the parameter block located at %P00300.

Rung 3 moves the new time value into the parameter block starting at R00300. It uses AND and ADD operations to retrieve the current clock value from %P00303 and replace the hours, minutes and seconds portion of the value with the values in NOON and MIN_SEC.

Rung 4 uses the parameter block beginning at %R00300 to set the new time.



SVC_REQ 8: Reset Watchdog Timer

Use SVC_REQ 8 to reset the watchdog timer during the scan.

Ordinarily, when the watchdog timer expires, the CPU shuts down without warning. SVC_REQ 8 allows the timer to keep going during a time-consuming task (for example, while waiting for a response from a communications line).

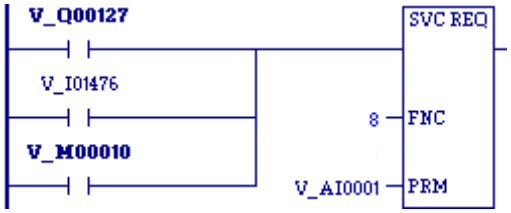
Warning

Be sure that resetting the watchdog timer does not adversely affect the controlled process.

SVC_REQ 8 has no associated parameter block; however, you must still specify a dummy parameter, which SVC_REQ 8 will not use.

Example

In the following LD example, power flow through enabling output %Q0127 or input %I1476 or internal coil %M00010 causes the watchdog timer to be reset.



SVC_REQ 9: Read Sweep Time from Beginning of Sweep

Use SVC_REQ 9 to read the time in milliseconds since the start of the sweep. The data format is unsigned 16-bit integer.

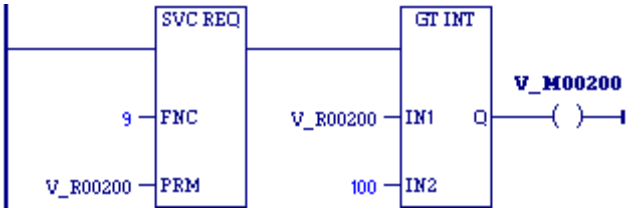
Output

The parameter block is an output parameter block only; it has a length of one word.

address	time since start of scan

Example

The elapsed time from the start of the scan is read into location %R00200. If it is greater than 100ms, internal coil %M0200 is turned on.



Note: Higher resolution (in nanoseconds) can be obtained by using SVC_REQ 51, described on page 10-50.

SVC_REQ 10: Read Target Name

Use SVC_REQ 10 to read the name of the currently executing target.

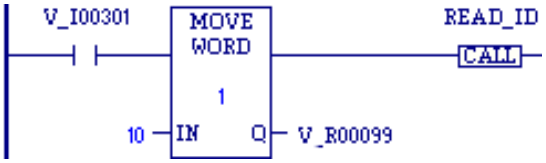
Output

The output parameter block has a length of four words. It returns eight ASCII characters: the target name (from one to seven characters) followed by null characters (00h). The last character is always a null character. If the target name has fewer than seven characters, null characters are appended to the end.

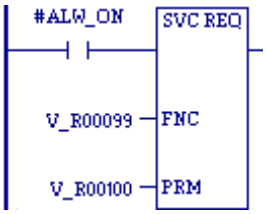
Address	Low Byte	High Byte
Address	character 1	character 2
Address+1	character 3	character 4
Address+2	character 5	character 6
Address+3	character 7	00

Example

When enabling input %I0301 goes OFF, register location %R0099 is loaded with the value 10, which is the function code for the Read Target Name function. The program block READ_ID is then called to retrieve the target name. The parameter block is located at address %R0100.



Program block READ_ID:



SVC_REQ 11: Read Controller ID

Use SVC_REQ 11 to read the name of the controller executing the program.

Output

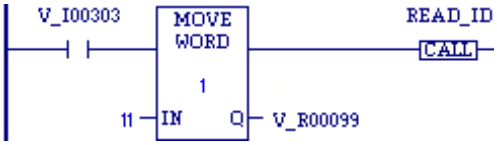
The output parameter block has a length of four words. It returns eight ASCII characters: the PLC ID (from one to seven characters) followed by null characters (00h). The last character is always a null character

If the PLC ID has fewer than seven characters, null characters are appended to the end.

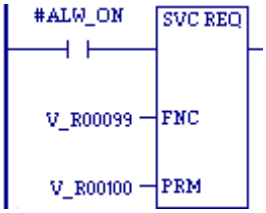
Address	Low Byte	High Byte
address	character 1	character 2
address+1	character 3	character 4
address+2	character 5	character 6
address+3	character 7	00

Example

When enabling input %I0303 is ON, register location %R0099 is loaded with the value 11, which is the function code for the Read PLC ID function. The program block READ_ID is then called to retrieve the ID. The parameter block is located at address %R0100.



Program Block READ_ID:



SVC_REQ 12: Read Controller Run State

Use SVC_REQ 12 to read the current RUN state of the CPU.

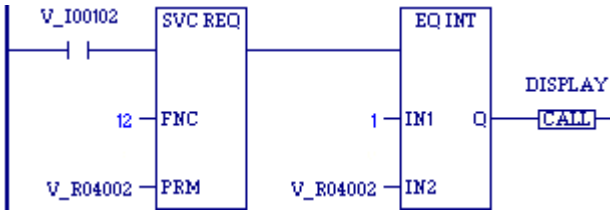
Output

The output parameter block has a length of one word.

address	1 = run/disabled
	2 = run/enabled

Example

When contact V_I00102 is ON, the CPU run state is read into location %R4002. If the state is Run/Disabled, the CALL function calls program block DISPLAY.



SVC_REQ 13: Shut Down (Stop) CPU

Use SVC_REQ 13 to stop the CPU after the specified number of scans has been performed. All outputs go to their designated default states at the start of the next CPU scan. An informational “Shut Down PLC” fault is placed in the Controller Fault Table. The I/O scan continues as configured.

SVC_REQ 13 has an input parameter block with a length of one word.

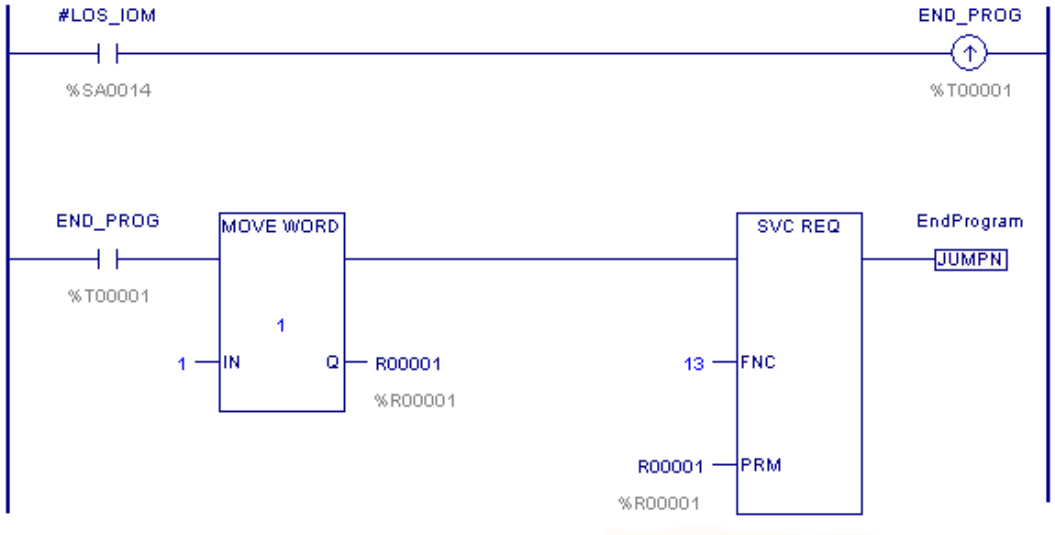
Address	Number of scans. Valid values: -1: The CPU uses the Number of Last Scans value configured in the Hardware Configuration Scan tab to determine when to transition to Stop mode. For details on Hardware Configuration parameters, refer to chapter 3. 1 through 5: The CPU finishes executing this scan, then executes this number of scans –1, and transitions to Stop mode.

Note: For CPUs with firmware version earlier than 2.00, the value must be set to 0; otherwise the CPU does not stop.

Example

When a “Loss of I/O Module” fault occurs, the #LOS_IOM contact turns ON and SVC_REQ 13 executes.

In this example, if the Shut Down CPU function executes, the JUMPN to the end of the program prevents the logic that follows the JUMPN from executing in the current sweep.



The block's last instruction is a LABELN:



SVC_REQ 14: Clear PLC or I/O Fault Table

Use SVC_REQ 14 to clear either the controller fault table or the I/O fault table. The SVC_REQ output is set ON unless some number other than 0 or 1 is entered as the requested operation.

The parameter block has a length of 1 word. It is an input parameter block only. There is no output parameter block.

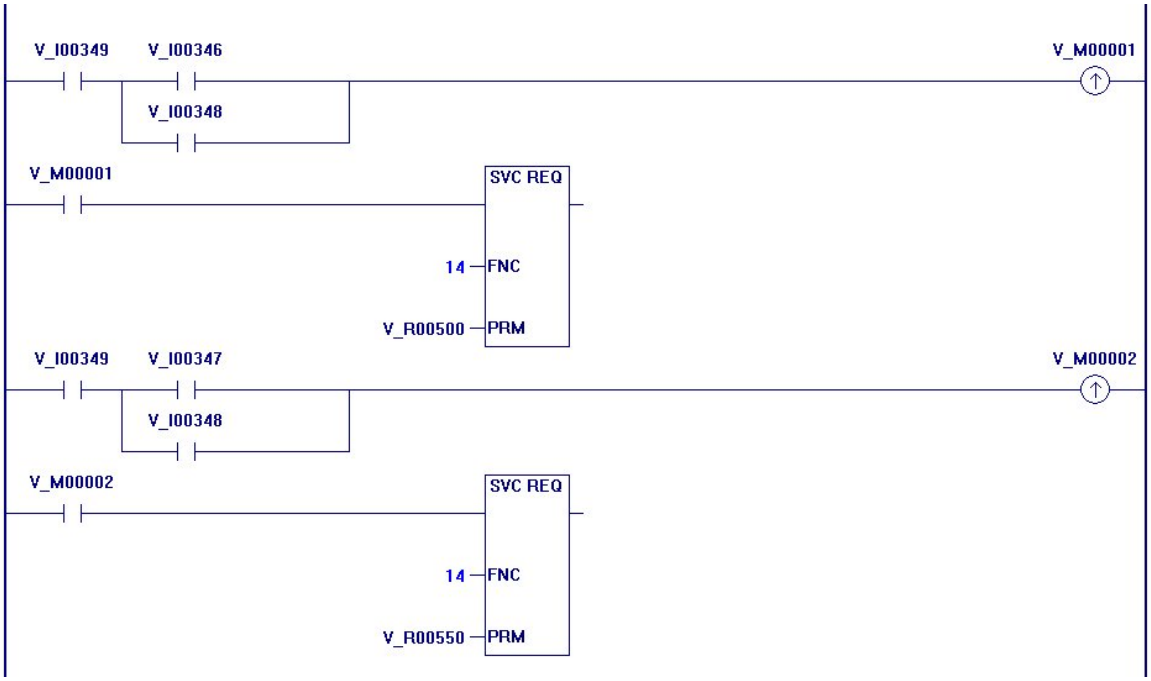
Address	0 = clear controller fault table
	1 = clear I/O fault table

Example

When inputs %I0346 and %I0349 are on, the controller fault table is cleared. When inputs %I0347 and %I0349 are on, the I/O fault table is cleared. When input %I0348 is on and input %I0349 is on, both are cleared. Positive transition coils V_M00001 and V_M00002 are used to trigger these service requests to prevent the fault tables from being cleared multiple times.

The parameter block for the controller fault table is located at %R0500; for the I/O fault table the parameter block is located at %R0550.

Note: Both parameter blocks are set up elsewhere in the program.



SVC_REQ 15: Read Last-Logged Fault Table Entry

Use SVC_REQ 15 to read the last entry logged in the controller fault table or the I/O fault table. The SVC_REQ output is set ON unless some invalid number is entered as the requested operation or the fault table is empty.

The non-extended parameter block has a length of 22 words and the extended parameter block has a length of 24 words.

Input Parameter Block

Address	***Format***
address+0	0 = Read controller fault table
	1 = Read I/O fault table
	80h = Read extended controller fault table
	81h = Read extended I/O fault table

Output Parameter Block

The format of the output parameter block depends on whether SVC_REQ 15 reads the controller fault table, the extended controller fault table, the I/O fault table or the extended I/O fault table.

Controller Fault Table Output Format		***Address***	***I/O Fault Table Output Format***	
High Byte	***Low Byte***		***High Byte***	***Low Byte***
	0	address+0		1
unused	long/short (always 01)	address+1	reference address memory type	long/short (always 03)
unused	unused	address+2	reference address offset	
slot	rack	address+3	slot	rack
	task	address+4	block	bus
fault action	fault group	address+5	point	
error code		address+6	fault action	fault category
fault extra data		address+7	fault type	fault category
		address+8 to address+18	fault extra data	fault description
minutes	seconds	address+19	minutes	seconds
day of month	hour	address+20	day of month	hour
year	month	address+21	year	month
milliseconds *(extended format only)*		address+22	milliseconds *(extended format only)*	
not used *(extended format only)*		address+23	not used *(extended format only)*	

Long/Short Value

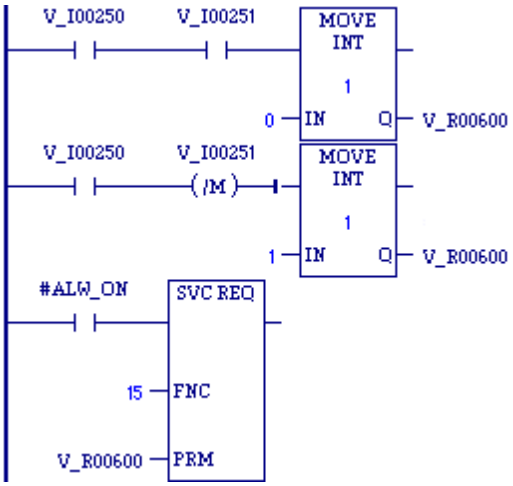
The first byte (low byte) of word *address +1* contains a number that indicates the length of the fault-specific data in the fault entry. Possible values are as follows:

PLC extended and non extended fault tables	00 = 8 bytes (short)	01 = 24 bytes (long)
I/O extended and non extended fault tables	02 = 5 bytes (short)	03 = 21 bytes (long)

Note: PACSystems CPUs always return the Long values for both extended and non-extended formats.

Example 1

When inputs %I0250 and %I0251 are both on, the first Move function places a zero (read controller fault table) into the parameter block for SVC_REQ 15. When input %I0250 is on and input %I0251 is off, the Move instruction instead places a one (read I/O fault table) in the SVC_REQ parameter block. The parameter block is located at location %R0600.



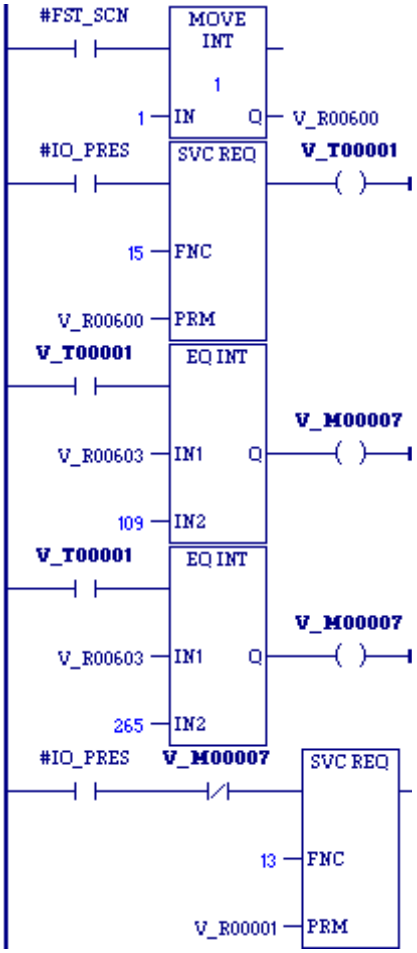
Example 2

The CPU is shut down when any fault occurs on an I/O module except when the fault occurs on modules in rack 0, slot 9 and in rack 1, slot 9. If faults occur on these two modules, the system remains running. The parameter for "table type" is set up on the first scan. The contact IO_PRES, when set, indicates that the I/O fault table contains an entry. The CPU sets the normally open contact in the scan after the fault logic places a fault in the table. If faults are placed in the table in two consecutive scans, the normally open contact is set for two consecutive scans.

The example uses a parameter block located at %R0600. After the SVC_REQ function executes, the second, third, and fourth words of the parameter block identify the I/O module that faulted:

	High Byte	***Low Byte***
%R0600		1
%R0601	reference address memory type	long/short
%R0602	reference address offset	
%R0603	slot number	rack number
%R0604	block (bus address)	I/O bus no.
%R0605		point address
%R0606	fault data	

In the program, the EQ_INT blocks compare the rack/slot address in the table to hexadecimal constants. The internal coil %M0007 is turned on when the rack/slot where the fault occurred meets the criteria specified above. If %M0007 is on, its normally closed contact is off, preventing the shutdown. Conversely, if %M0007 is off because the fault occurred on a different module, the normally closed contact is on and the shutdown occurs.



SVC_REQ 16: Read Elapsed Time Clock

Use SVC_REQ 16 to read the system's elapsed time clock. The elapsed time clock measures the time in seconds since the CPU was powered on. The parameter block has a length of three words used for output only.

Output

address	Seconds from power on (low order)
address+1	Seconds from power on (high order)
address+2	100 microsecond ticks

The first two words are the elapsed time in seconds. The last word is the number of 100 microsecond ticks in the current second.

The resolution of the PLC's elapsed time clock is 100 microseconds. The overall accuracy of the elapsed time clock is ± 0.01%. The accuracy of an individual sample of the elapsed time clock is approximately 105 microseconds.

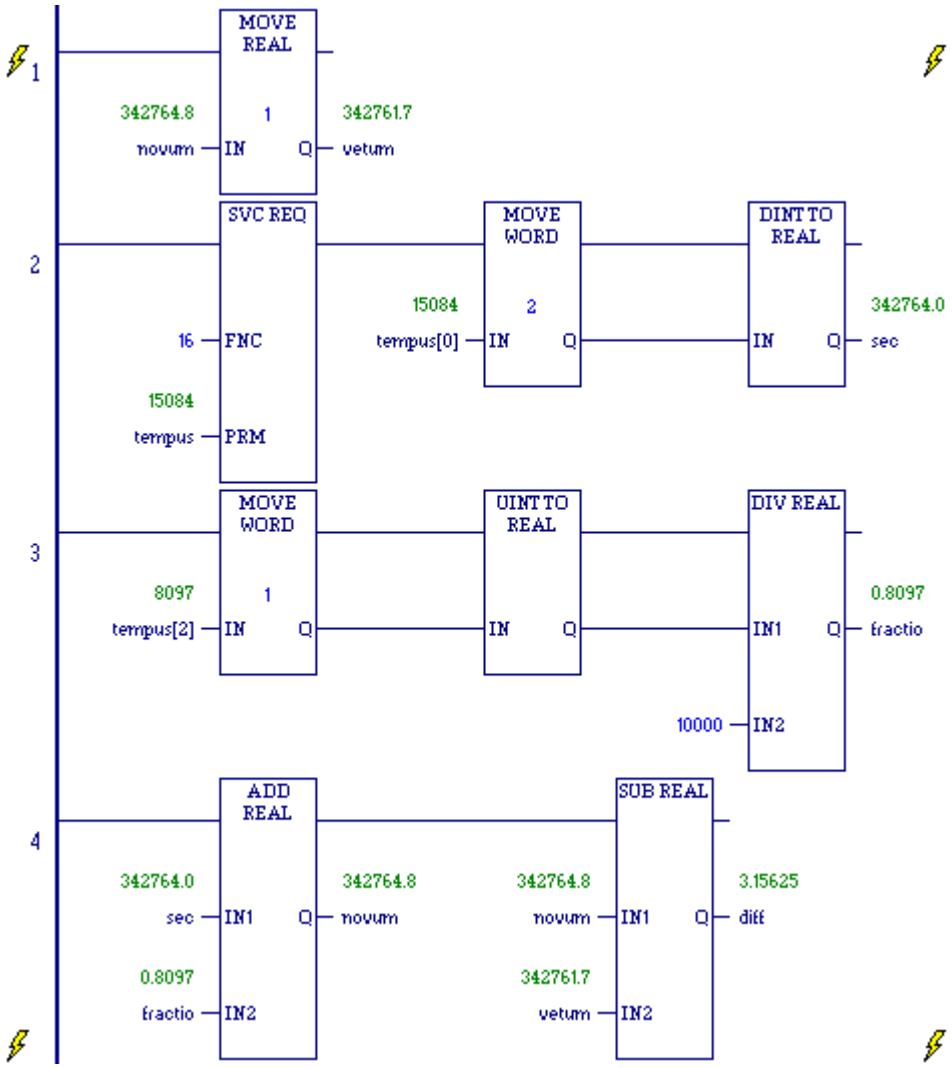
Warning

The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 16 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic

Example

The following logic is used in a block that is called infrequently. The screen shot was taken between calls to the block. The logic displayed calculates the number of seconds that have elapsed since the last time the block was called. It performs the final operation on rung 4 by subtracting the time obtained by SVC_REQ 16 the last time the block was called (vetum) from the time currently obtained by SVC_REQ 16 (novum) and storing the calculated value in the variable named diff.

On rung 2, SVC_REQ 16 returns three WORDs, stored in the 3-WORD array tempus. The first two WORDs (16-bit values) are moved to a DINT (a 32-bit value). This move amounts to a rough data type conversion that ignores the fact that the DINT type is actually a signed value. Despite that, the subsequent calculations are correct until the time since power-on reaches approximately 50 years. The DINT is converted to REAL to yield the number of whole seconds elapsed since power-on, stored in variable sec. On rung 3, the third word returned by SVC_REQ 16, tempus[2], is converted to REAL. This is the number of 100 microsecond ticks. To obtain a fraction of a second, stored in the variable fractio, the value is divided by 10,000. On rung 4, sec and fractio are added to express the exact number of seconds elapsed since power-on, and this value is stored in the variable novum. On rung 1, the previous value of novum was saved as vetum, the exact number of seconds elapsed since power-on the last time the block was called. The last instruction on the fourth rung subtracts vetum from novum to yield the number of seconds that have elapsed since the last time the block was called.



Note: Higher resolution (in nanoseconds) can be obtained by using SVC_REQ 50, described on page 10-47.

SVC_REQ 17: Mask/Unmask I/O Interrupt

Use SVC_REQ 17 to mask or unmask an interrupt from an input/output board. When an interrupt is masked, the CPU does not execute the corresponding interrupt block when the input transitions and causes an interrupt.

The parameter block is an input parameter block only; it has a length of three words.

address	0 = unmask input 1 = mask input
address+1	memory type
address+2	reference (offset)

"Memory type" is a decimal number that resides in the low byte of word *address + 1*. It corresponds to the memory type of the input:

70	%I memory in bit mode
10	%AI memory
12	%AQ memory

Successful execution occurs unless:

- Some number other than 0 or 1 is entered as the requested operation.
- The memory type of the input/output to be masked or unmasked is not %I, %AI or %AQ memory.
- The I/O board is not a supported input/output module.
- The reference address specified does not correspond to a valid interrupt trigger reference.
- The specified channel does not have its interrupt enabled in the configuration.

Masking/Unmasking Module Interrupts

During module configuration, interrupts from a module can be enabled or disabled. If a module's interrupt is disabled, it cannot be used to trigger logic execution in the application program and it cannot be unmasked. However, if an interrupt is enabled in the configuration, it can be dynamically masked or unmasked by the application program during system operation.

The application program can mask and unmask interrupts that are enabled using Service Request Function Block #17. To mask or unmask an interrupt from a non-GE Fanuc VME module, the application logic should pass VME_INT_ID (17 decimal, 11H) as the memory type and the VME interrupt id as the offset to SVC_REQ 17.

When the interrupt is not masked, the CPU processes the interrupt and schedules the associated program logic for execution. When the interrupt is masked, the CPU processes the interrupt but does not schedule the associated program logic for execution.

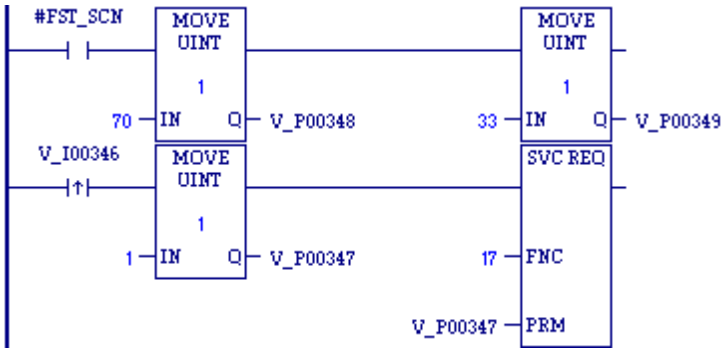
When the CPU transitions from Stop to Run, the interrupt is unmasked.

For additional information on configuring and using VME module interrupts in a PACSystems RX7i control system, refer to *PACSystems RX7i User's Guide to Integration of VME Modules,* GFK-2235.

Example 1

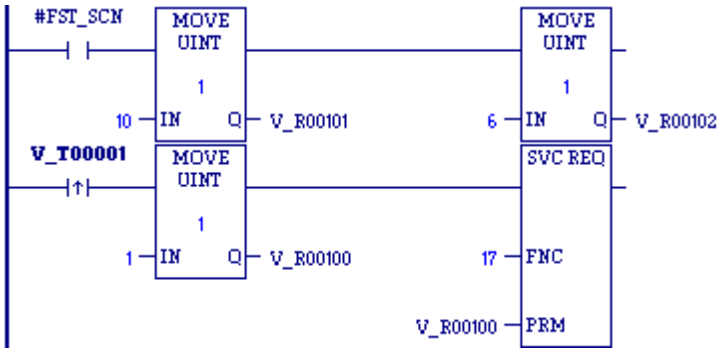
In this example, interrupts from input %I00033 are masked. The following values are moved into the parameter block, which starts at %P00347, on the first scan:

address	%P00347	1	Interrupts from input are masked.
address + 1	%P00348	70	Input type is %I.
address + 2	%P00349	33	Offset is 33.



Example 2

When %T00001 transitions on, alarm interrupts from input %AI0006 are masked. The parameter block at %R00100 is set up on the first scan.



SVC_REQ 18: Read I/O Forced Status

Use SVC_REQ 18 to read the current status of forced values in the CPU's %I and %Q memory areas.

Note: SVC_REQ 18 does not detect overrides in %G or %M memory types. Use %S0011 (#OVR_PRE) to detect overrides in %I, %Q, %G, %M, and symbolic memory types.

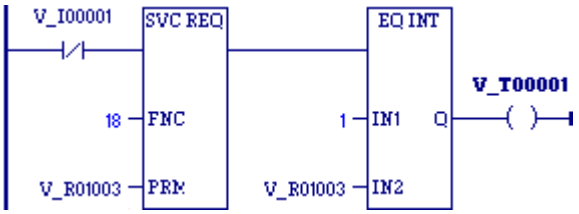
The parameter block has a length of one word used for output only.

Output

address	0 = No forced values are set
	1 = Forced values are set

Example

SVC_REQ reads the status of I/O forced values into location %R1003. If the returned value in %R1003 is 1, there is a forced value, and EQ INT turns the %T0001 coil ON.



SVC_REQ 19: Set Run Enable/Disable

Use SVC_REQ 19 to permit the LD program to control the RUN mode of the CPU.

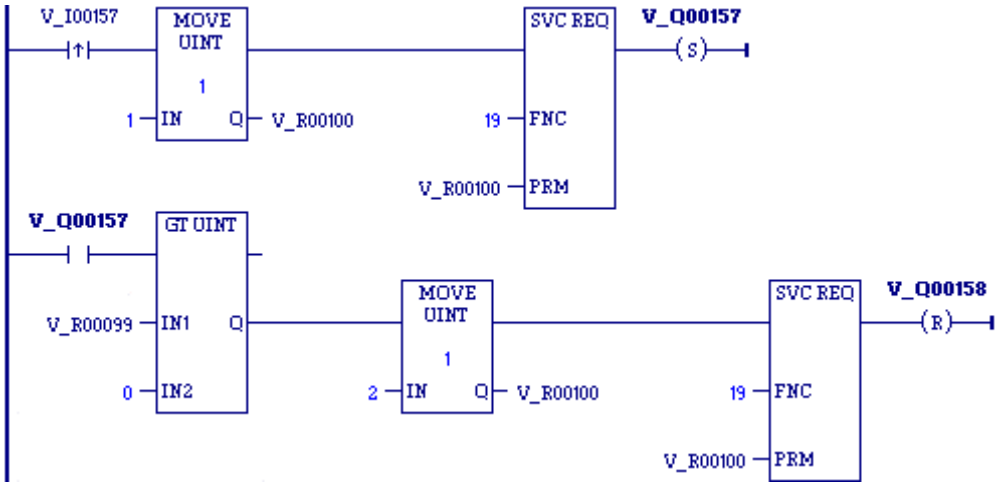
The parameter passed indicates which function to perform. The OK output is turned ON if the function executes successfully. It is set OFF if the requested operation is not SET RUN DISABLE mode (1) or SET RUN ENABLE mode (2).

The parameter block is an input parameter block only with this format:

address	1 = SET RUN DISABLE mode 2 = SET RUN ENABLE mode

Example

When input %I00157 transitions to on, the RUN DISABLE mode is set. When the SVC_REQ function successfully executes, coil %Q00157 is turned on. When %Q00157 is on and register %R00099 is greater than zero, the mode is changed to RUN ENABLE mode. When the SVCREQ successfully executes, coil %Q00157 is turned off.



SVC_REQ 20: Read Fault Tables

Use SVC_REQ 20 to retrieve the entire PLC or I/O fault table and return it to the LD program in designated registers.

The first input parameter designates which table is to be read. A second input parameter (always zero for the standard Read Fault Tables) is used by the extended format to read a designated fault entry or to read a range of fault entries. The fault table data is placed in the parameter block following the input parameters.

The OK output is turned on if the function executes successfully. It is off if the requested operation is not Read Controller Fault Table (00h), Read I/O Fault Table (01h), Read Extended Controller Fault Table (80h), or Read Extended I/O Fault Table (81h), or if there is not enough of the specified memory reference to hold the fault data. If the specified fault table is empty, the function sets the OK output on, but returns only the fault table header information.

The parameter block is an input and output parameter block. The parameter block comes in two formats:

- Non-Extended: Read Controller Fault Table (00h), Read I/O Fault Table (01h)
- Extended: Read Extended Controller Fault Table (80h), Read Extended I/O Fault Table (81h)

Non-Extended Formats

For non-extended formats, SVC_REQ 20 requires 693 registers available.

Input Parameter Block Format

address + 0	00h = Read Controller fault table 01h = Read I/O fault table
address + 1	Always 0

Non-Extended Output Parameter Block Format

Controller Fault Table Output Format		***Address***	***I/O Fault Table Output Format***	
High Byte	***Low Byte***		***High Byte***	***Low Byte***
Unused	0 = Controller Fault Table	**address+0**	Unused	1 = I/O Fault Table
Unused	Always zero (0)	**address+1**	Unused	Always zero (0)
Unused	Unused	**address+2**	Unused	Unused
Unused	Unused	**address+3— address+14**	Unused	Unused
Minutes	Seconds	**address+15— address+17 (time since last clear, in BCD format)**	Minutes	Seconds
Day Of Month	Hour		Day of month	Hour
Year	Month		Year	Month
Number of faults since last clear		**address+18**	Number of faults since last clear	
Number of faults in queue		**address+19**	Number of faults in queue	
Number of faults read		**address+20**	Number of faults read	
Start of fault data		**address+21**	Start of fault data	

For the non-extended formats, the returned data for each fault consists of 21 words (42 bytes). This request returns 16 controller fault table entries or 32 I/O fault table entries, or the actual number of faults if it is fewer. If the fault table read is empty, no data is returned.

The following table shows the return format of a controller fault table entry and an I/O fault table entry.

Format of Returned Data for Fault Table Entries

Controller Fault Table Output Format		***Address***	***I/O Fault Table Output Format***	
High Byte	***Low Byte***		***High Byte***	***Low Byte***
Unused	Long/short	**address+21**	Memory type	Long/Short*
Unused	Unused	**address+22**	Offset	
Slot	Rack	**address+23**	Slot	Rack
Task		**address+24**	Bus address	I/O Bus Number (block)
Fault action	Fault group	**address+25**	Point	
Error code		**address+26**	Fault action	Fault group
Fault extra data		**address+27**	Fault type	Fault category
		address+28	Fault extra data	Fault description
		address+29—address+38	Fault extra data	
Minutes	Seconds	**address+39—address+41 (time stamp, in BCD format)**	Minutes	Seconds
Day of month	Hour		Day of month	Hour
Year	Month		Year	Month
Start of next fault output parameter block		**address+42**	Start of next fault output parameter block	

* The Long/Short indicator in the low byte of *Address + 21* specifies the amount of fault data present in the fault entry:

Fault Table	***Long/Short Value***	***Fault Data Returned***
PLC	00	8 bytes of fault extra data present in the fault entry
	01	24 bytes of fault extra data
I/O	02	5 bytes of fault extra data
	03	21 bytes of fault extra data

Extended Formats

Each extended format request can read a maximum of 64 faults, or the size of the fault table if it contains less than 64 faults.

For extended formats (Read Extended Controller Fault Table (80h), or Read Extended I/O Fault Table (81h)), the PLC calculates the number of entries being read. You must ensure that enough registers are available to receive the amount of fault entries requested. If the amount of data requested exceeds the registers available, the CPU returns a fault indicating that reference memory is out of range.

The total size of the fault table for the extended fault format is

Header Size + ((# fault entries) * (size of fault entry))

Input Parameter Block Format

address+0	80h = Read extended controller fault table 81h = Read extended I/O fault table
address+1	Starting index of faults to be read
address+2	Number of faults to be read

Extended Format Output Parameter Block Format

Controller Fault Table Output Format		***Address***	***I/O Fault Table Output Format***	
High Byte	***Low Byte***		***High Byte***	***Low Byte***
Unused	80h = Extended controller fault table	**address**	Unused	81h = Extended I/O fault table
Starting index of faults to be read		**address+1**	Starting index of faults to be read	
Number of faults to be read		**address+2**	Number of faults to be read	
Unused	Unused	**address+3—address+14**	Unused	Unused
Minutes	Seconds	**address+15—address+17 (time since last clear, in BCD format)**	Minutes	Seconds
Day of Month	Hour		Day of month	Hour
Year	Month		Year	Month
Number of faults since last clear		**address+18**	Number of faults since last clear	
Number of faults in queue		**address+19**	Number of faults in queue	
Number of faults read		**address+20**	Number of faults read	
Unused		**address+21—address+36**	Unused	
Start of fault data		**address+37**	Start of fault data	

For Read Extended Controller Fault Table (80h) and Read Extended I/O Fault Table (81h), the returned data for each fault entry consists of 23 words (46 bytes).

Format of Returned Data for Fault Table Entries

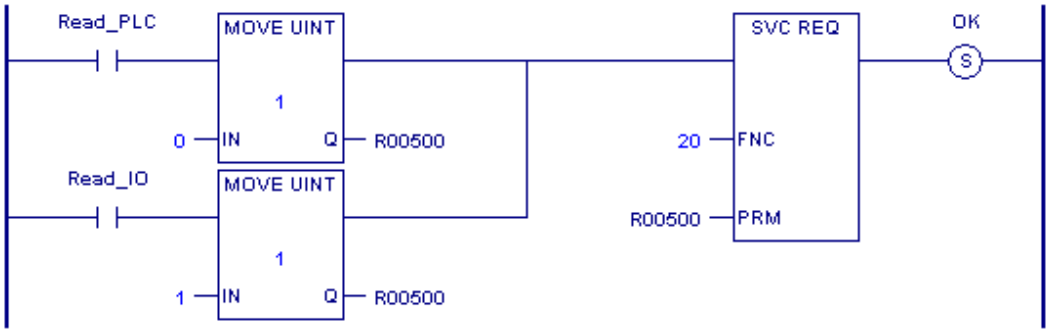
Controller Fault Table Output Format		***Address***	***I/O Fault Table Output Format***	
High Byte	***Low Byte***		***High Byte***	***Low Byte***
Unused	Long/Short	**address+37**	Reference address memory type	Long/Short (See page 10-34.)
Unused	Unused	**address+38**	Reference address offset	
Slot	Rack	**address+39**	Slot	Rack
Task		**address+40**	Bus address	I/O bus number (block)
Fault action	Fault group	**address+41**	point	
Error code		**address+42**	Fault action	Fault group
Fault extra data		**address+43**	Fault type	Fault category
		address+44	Fault extra data	Fault description
		address+45— address+54	Fault extra data	
Minutes	Seconds	**address+55— address+58 (time stamp in BCD format)**	Minutes	Seconds
Day of month	Hour		Day of month	Hour
Year	Month		Year	Month
Milliseconds			Milliseconds	
Not used		**address+59**	Not used	

Start of next fault output parameter block	**address+60**	Start of next fault output parameter block

SVC_REQ 20 Examples

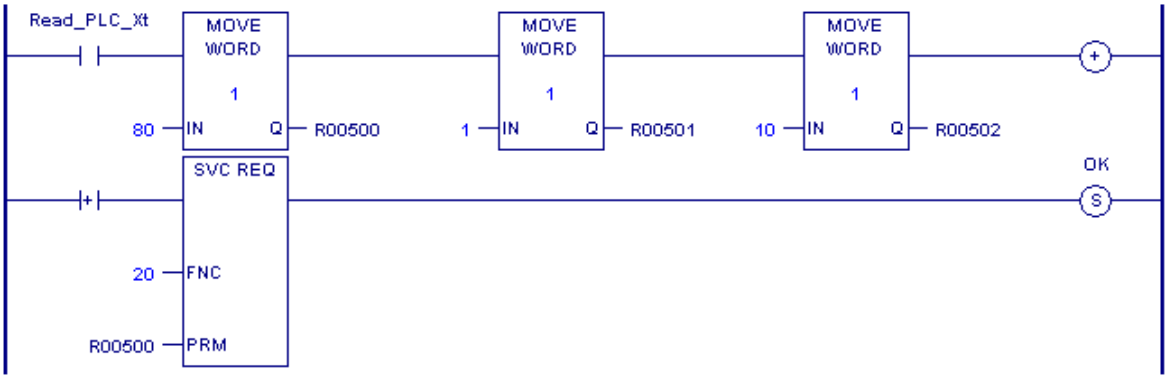
Example 1: Non-Extended Format

When Read_PLC transitions on, a value of 0 is moved to the parameter block, which is located at %R00500, and the controller fault table is read. When Read_IO transitions on, a value of 1 is moved to the parameter block and the I/O fault table is read. When the SVC_REQ function successfully executes, coil OK is turned on.



Example 2: Extended Format

When Read_PLC_Xt transitions on, the Extended controller fault table is read. The parameter block begins at %R00500. %R00500 contains the fault table type (PLC Extended); %R00501 contains the starting fault to read, and %R00502 contains the number of faults to read starting with the fault number in %R00501. When the SVC_REQ function successfully executes, coil OK is turned on.



SVC_REQ 21: User-Defined Fault Logging

Use SVC_REQ 21 to define a fault that can be displayed in the controller fault table. The fault contains binary information or an ASCII message. The user-defined fault codes start at 0 hex.

The error code information for the fault must be within the range 0 to 2047 for an "Application Msg:" to be displayed. If the error code is in the range 81 to 112 decimal, the CPU sets a fault bit of the same number in %SA system memory. This allows up to 32 bits to be individually set.

Error Code	*Status Bit*
Errors 0—80	No bit set
Errors 81—112	Sets %SA
Errors 113—2047	No bit set
Errors 2048—32,767	Reserved

When EN is active, the fault data array referenced by IN is logged as a fault to the controller fault table. If EN is not enabled, the ok bit is cleared. If the error code is out of range, the ok bit is cleared and the fault will not be logged as requested.

The parameter block is an input parameter block only with this format:

	Error code	
Parameter address	*MSB*	*LSB*
address+1	Text2	Text1
address+2	Text4	Text3
address+3	Text6	Text5
address+4	Text8	Text7
address+5	Text10	Text9
address+6	Text12	Text11
address+7	Text14	Text13
address+8	Text16	Text15
address+9	Text18	Text17
address+10	Text20	Text19
address+11	Text22	Text21
address+12	Text24	Text23

The input parameter data allows you to select an error code in the range 0 to 2047 and text information that will be placed in the fault extra data portion of a long PLC fault. The PLC fault address, fault group, and fault action are filled in by the function block.

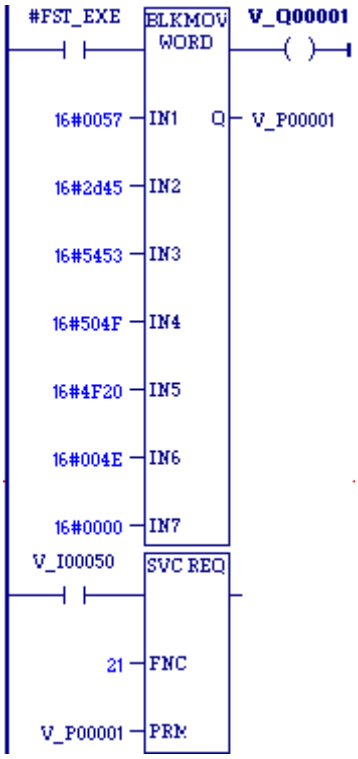
The fault text bytes 1 – 24 can be used to pass binary or ASCII data with the fault. If the first byte of the fault text data is non-zero, the data will be an ASCII message string. This message will then be displayed in the fault description area of the fault table. If the message is less than 24 characters, the ASCII string must be NULL byte-terminated. The programmer will display "Application Msg:" and the ASCII data will be displayed as a message immediately following "Application Msg:". If the error code is between 1 and 2047, the error code number will be displayed immediately after "Msg" in the "Application Msg:" string. (If the error code is greater than 2047, the function is ignored and its output is set to OFF.)

If the first byte of text is zero, then only “Application Msg:” will display in the fault description. The next 1-23 bytes will be considered binary data for user data logging. This data is displayed in the controller fault table.

Note: When a user-defined fault is displayed in the controller fault table, a value of -32768 (8000 hex) is added to the error code. For example, the error code 5 will be displayed as -32763.

Example

The value passed to IN1 is the fault error code. The value passed in, 16x0057, represents an error code of 87 decimal and will appear as part of the fault message. The values of the next inputs give the ASCII codes for the text of the error message. For IN2, the input is 2D45. The low byte, 45, decodes to the letter **E** and the high byte, 2D, decodes to -. Continuing in this manner, the string continues with **S T O P O** and **N**. The final character, **00**, is the null character that terminates the string. In summary, the decoding yields the string message **E_STOP ON**.



SVC_REQ 22: Mask/Unmask Timed Interrupts

Use SVC_REQ 22 to mask or unmask timed interrupts and to read the current mask. When the interrupts are masked, the CPU does not execute any timed interrupt block timed program that is associated with a timed interrupt. Timed interrupts are masked/unmasked as a group. They cannot be individually masked or unmasked.

Successful execution occurs unless some number other than 0 or 1 is entered as the requested operation or mask value.

The parameter block is an input and output parameter block.

To determine the current mask, use this format:

address	0 = Read interrupt mask

The CPU returns this format:

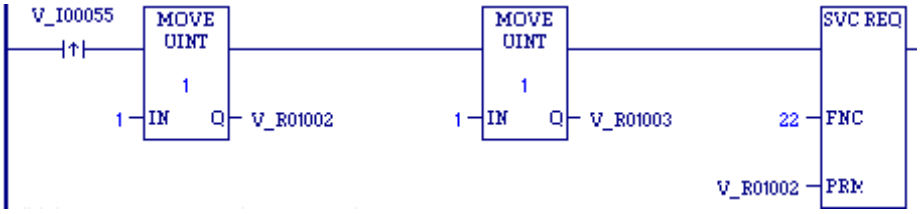
address	0 = Read interrupt mask
address+1	0 = Timed interrupts are unmasked 1 = Timed interrupts are masked

To change the current mask, use this format:

address	1 = Mask/unmask interrupts
address+1	0 = Unmask timed interrupts 1 = Mask timed interrupts

Example

When input %I00055 transitions on, timed interrupts are masked.



SVC_REQ 23: Read Master Checksum

Use SVC_REQ 23 to read master checksums for the set of user program(s) and the configuration, and to read the checksum for the block from which the service request is made.

There is no input parameter block for this service request. The output parameter block requires 15 words of memory.

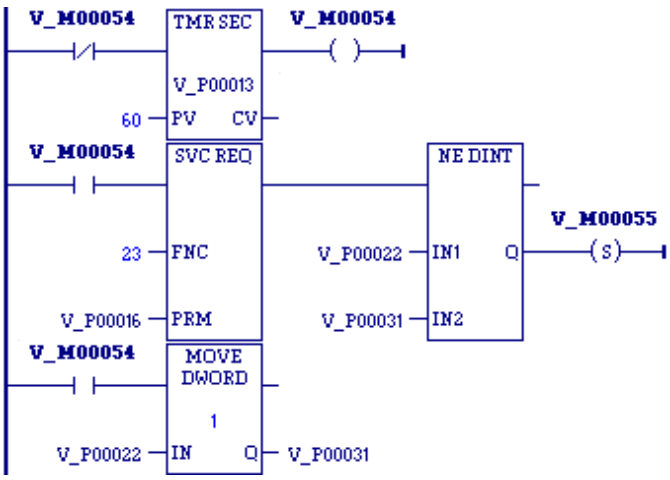
Output

When a RUN MODE STORE is active, the program checksums may not be valid until the store is complete. To determine when checksums are valid, three flags (one each for Program Block Checksum, Master Program Checksum, and Master Configuration Checksum) are provided at the beginning of the output parameter block.

Address	Description
Address	Program Checksum Valid (0 = not valid, 1 = valid)
Address + 1	Master Program Checksum Valid (0 = not valid, 1 = valid)
Address + 2	Master Configuration Checksum Valid (0 = not valid, 1 = valid)
Address + 3	Number of LD/SFC Blocks (including _MAIN)
Address + 4	Size of User Program in Bytes (DWORD data type)
Address + 6	Program Set Additive Checksum
Address + 7	Program CRC Checksum (DWORD data type)
Address + 9	Size of Configuration Data in Kbytes
Address + 10	Configuration Additive Checksum
Address + 11	Configuration CRC Checksum (DWORD data type)
Address + 13	high byte: always zero low byte: Currently Executing Block's Additive Checksum
Address + 14	Currently Executing Block's CRC Checksum

Example – SVC_REQ 23

When the timer using registers %P00013 through %P00015 expires, the checksum read is performed. The checksum data returns in registers %P00016 through %P00030. The master program checksum in registers %P00022 and %P00023 (the program checksum is a DWORD data type and occupies two adjacent registers) is compared with the last saved master program checksum. If these are different, coil %M00055 is latched on. The current master program checksum is then saved in registers %P00031 and %P00032.



SVC_REQ 24: Reset Module

Use SVC_REQ 24 to reset a daughterboard or some modules. Modules that support SVC_REQ 24 include:

RX3i IC693BEM331, IC694BEM331, IC693APU300, IC694APU300, IC695ETM001, IC693ALG2222, IC694ALG2222

RX7i: Embedded Ethernet Interface module, IC697BEM731, IC698BEM731, IC697HSC700, IC697ALG230, IC698ETM001

The SVCREQ output is set ON unless one of the following conditions exists:

-An invalid number for rack and/or slot is entered.

-There is no module at the specified location.

-The module at the specified location does not support a runtime reset.

-The CPU was unable to reset the module at the specified location.

For this function, the parameter block has a length of 1 word. It is an input parameter block only.

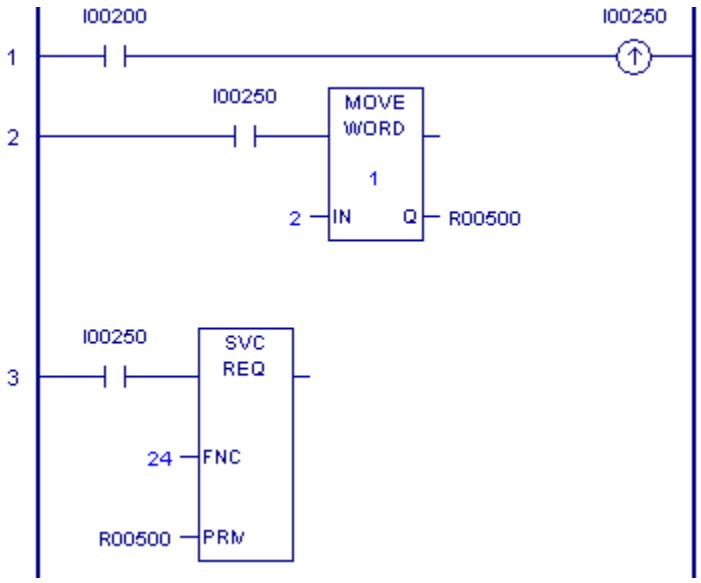
address	Module slot (low byte) Module rack (high byte) *Rack 0, Slot 1 indicates that a reset is to be sent to the daughterboard.*

Note: It is important to invoke SVC_REQ #24 for a given module for only one sweep at a time. Each time this function executes, the target module will be reset regardless of whether it has finished starting up from a previous reset.

After sending a SVC_REQ #24 to a module, you must wait a minimum of 5 seconds before sending another SVC_REQ #24 to the same module. This ensures that the module has time to recover and complete its startup.

Example

This example resets the module in rack 0/slot 2. In rung 1, when contact %I00200 is closed, the positive transition coil sets %I00250 to ON for one sweep. The MOVE_WORD instruction in rung 2 receives power flow and moves the value 2 into %R00500. The SVC_REQ function in rung 3 then receives power flow and resets the module indicated by the rack/slot value in %R00500.



SVC_REQ 25: Disable/Enable EXE Block and Standalone C Program Checksums

Use SVC_REQ 25 to enable or disable the inclusion of EXE in the background checksum calculation. The default is to include the checksums.

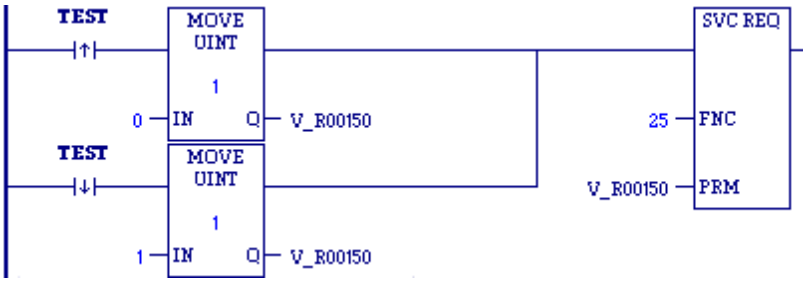
This service request uses only an input parameter block.

address	0 = Disable C applications inclusion in checksum calculation
	1 = Enable C application inclusion in checksum calculation

The parameter block is unchanged after execution of the service request.

Example

When the coil TEST transitions from OFF to ON, SVC_REQ 25 executes to disable the inclusion of EXE blocks in the background checksum calculation. When coil TEST transitions from ON to OFF, the SVC_REQ executes to again include EXE blocks in the background checksum calculation.



SVC_REQ 29: Read Elapsed Power Down Time

Use SVCREQ 29 to read the amount of time elapsed between the last power-down and the most recent powerup. If the watchdog timer expired before power-down, the PLC is not able to calculate the power down elapsed time, so the time is set to 0.

This service request cannot be accessed from a C block.

This function has an output parameter block only. The parameter block has a length of three words.

address	Power-down elapsed seconds (low order)
address + 1	Power-down elapsed seconds (high order)
address + 2	100µS ticks

The first two words are the power-down elapsed time in seconds. The last word is the number of 100 microsecond ticks in the current second.

Note: Although this request responds with a resolution of 100µS, the actual accuracy is 1 second. The battery-backed clock used when the PLC is powered down is accurate to within 1 second.

Example of SVCREQ 29

When input %I0251 is ON, the elapsed power-down time is placed into the parameter block that starts at %R0050. The output coil (%Q0001) is turned on.



SVC_REQ 32: Suspend/Resume I/O Interrupt

Use SVC_REQ 32 to suspend a set of I/O interrupts and cause occurrences of these interrupts to be queued until these interrupts are resumed. The number of I/O interrupts that can be queued depends on the I/O module's capabilities. The CPU informs the I/O module that its interrupts are to be suspended or resumed. The I/O module's default is resumed. The Suspend applies to all I/O interrupts associated with the I/O module. Interrupts are suspended and resumed within a single scan.

SVC_REQ 32 uses only an input parameter block. Its length is three words.

Address	0 = resume interrupt 1 = suspend interrupt
Address + 1	memory type
Address + 2	reference (offset)

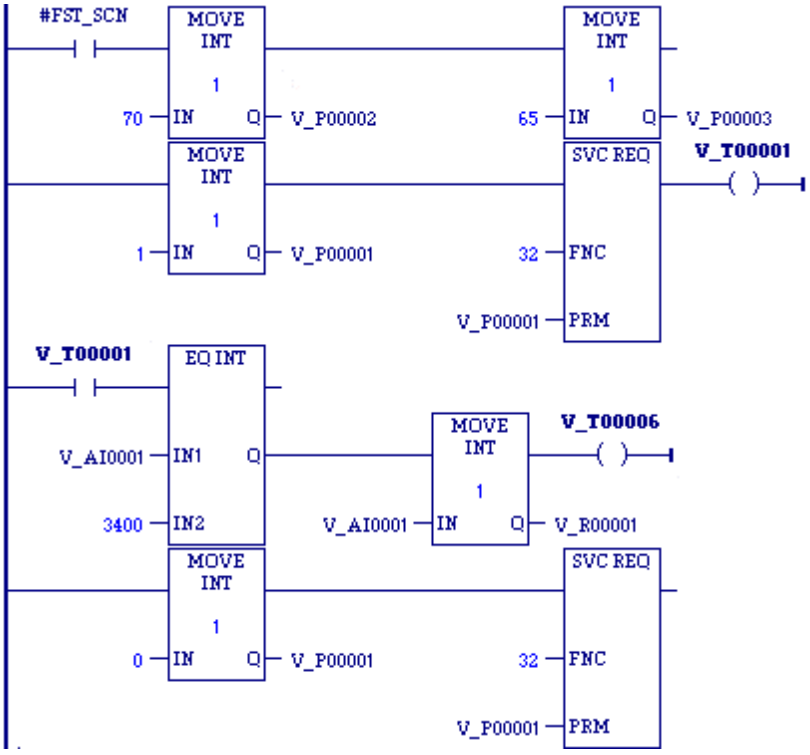
Successful execution occurs unless:

- Some number other than 0 or 1 is passed in as the first parameter.
- The memory type parameter is not 70 (%I memory).
- The I/O module associated with the specified address is not an appropriate module for this operation.
- The reference address specified is not the first %I reference for the High Speed Counter.
- Communication between the CPU and this I/O module has failed. (The board is not present, or it has experienced a fatal fault.)

Example – SVC_REQ 32

Interrupts from the high speed counter module whose starting point reference address is %I00065 will be suspended while the CPU solves the logic of the second rung. Without the Suspend, an interrupt from the HSC could occur during execution of the third rung and %T00006 could be set while %R000001 has a value other than 3,400. (%AI00001 is the first non-discrete input reference for the High Speed Counter.)

Note: I/O interrupts, unless suspended or masked, can interrupt the execution of a function block. The most often used application of this Service Request is to prevent the effects of the interrupts for diagnostic or other purposes.



SVC_REQ 45: Skip Next I/O Scan

Use the SVCREQ function #45 to skip the next output and input scans. Any changes to the output reference tables during the sweep in which the SVCREQ #45 was executed will not be reflected on the physical outputs of the corresponding modules. Any changes to the physical input data on the modules will not be reflected in the corresponding input references during the sweep after the one in which the SVCREQ #45 was executed.

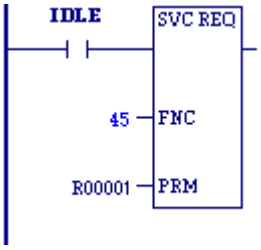
This function has no parameter block.

Note: This service request is provided for conversion of Series 90-30 applications. The Suspend I/O (SUS_IO) function block, which is supported by all PACSystems firmware versions, should be used in new applications.

Note: The DOIO Function Block is not affected by the use of SVCREQ #45. It will still update the I/O when used in the same logic program as the SVCREQ #45.

Example

In the following LD example, when the “Idle” contact passes power flow, the next Output and Input Scan are skipped.



SVC_REQ 50: Read Elapsed Time Clock

Use SVC_REQ 50 to read the system's elapsed time clock. The elapsed time clock measures the time in seconds since the CPU was powered on. The parameter block has a length of four words used for output only.

Output

address	Seconds from power on (low order)
address+1	Seconds from power on (high order)
address+2	nanosecond ticks (low order)
address+3	nanosecond ticks (high order)

The first two words are the elapsed time in seconds. The second two words are the number of nanoseconds elapsed in the current second.

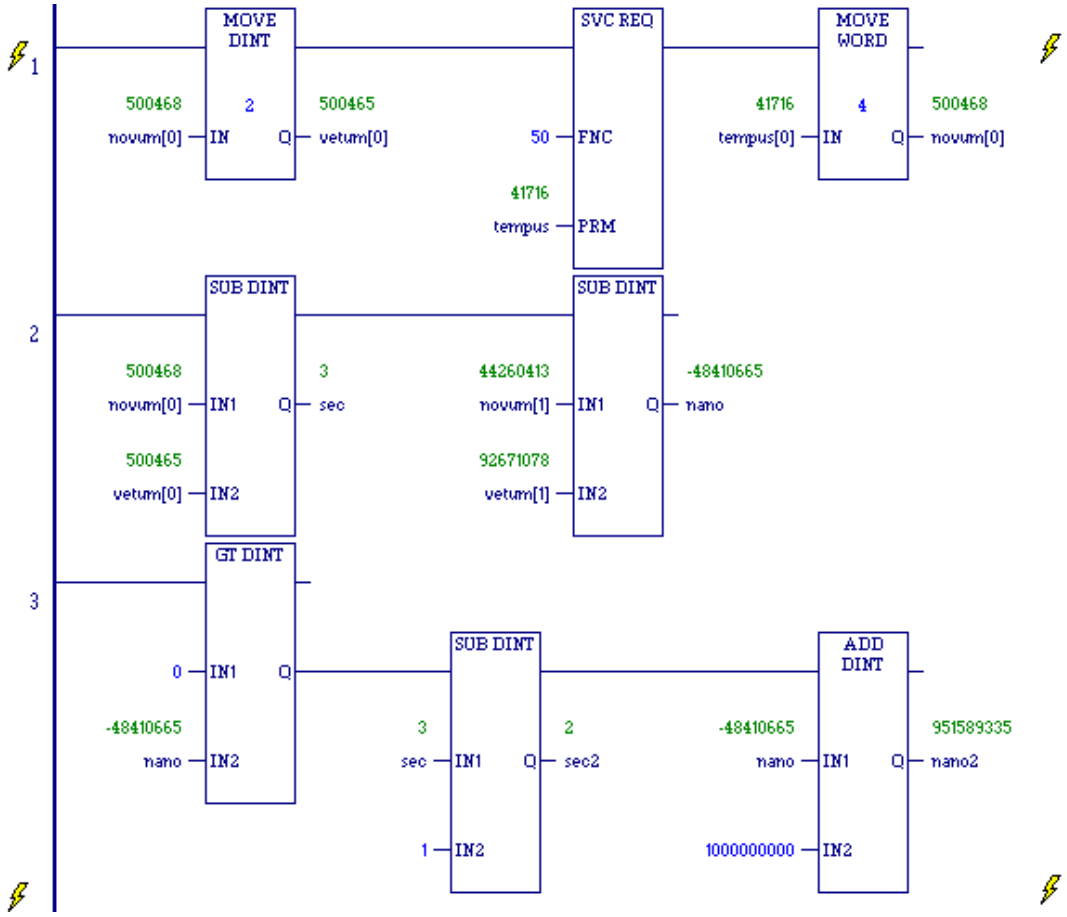
The resolution of the PLC's elapsed time clock is 100 microseconds. The overall accuracy of the elapsed time clock is ± 0.01%. The accuracy of an individual sample of the elapsed time clock is approximately 105 microseconds.

Warning

The SVC_REQ instruction is not protected against operating system and user interrupts. The timing and length of these interrupts are unpredictable. The clock sample returned by SVC_REQ 50 can sometimes be much more than 105 microseconds old by the time execution is returned to the LD logic

Example – SVC_REQ 50

The following logic is used in a block that is called once in a while. The screen shot was taken between calls to the block. The second rung of logic calculates the number of seconds that have elapsed since the last time the block was called. The third rung calculates the number of nanoseconds to be added to, or subtracted from, the number of seconds. The first rung saves the previous value of novum[0] and novum[1] into vetum[0] and vetum[1] before the second rung of logic places the current time values in novum[0] and novum[1].



SVC_REQ 51: Read Sweep Time from Beginning of Sweep

Use SVC_REQ 51 to read the time in nanoseconds since the start of the sweep. The data is unsigned 32-bit integer.

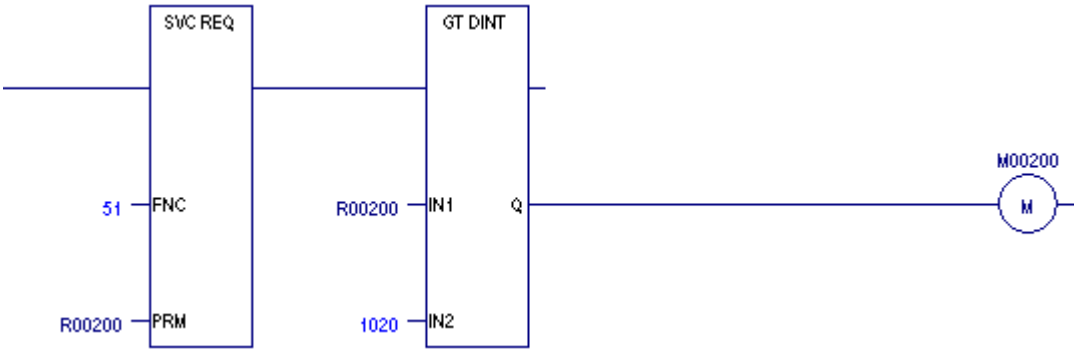
Output

The parameter block is an output parameter block only; it has a length of two words.

address	time (nanoseconds) since start of scan – low order
address+1	time (nanoseconds) since start of scan – high order

Example

The elapsed time from the start of the scan is read into locations %R00200 and %R00201 if it is greater than 10,020ns, internal coil %M0200 is turned on.

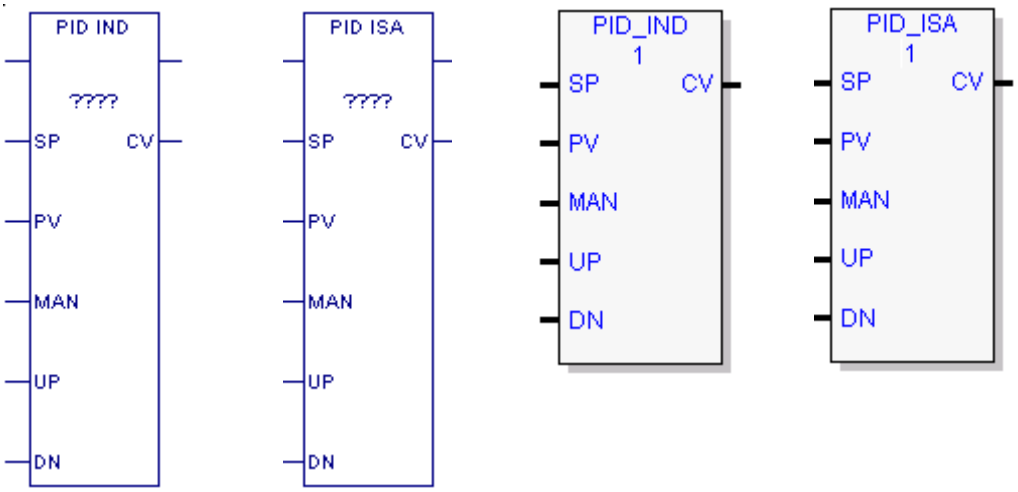


Chapter 11

PID Built-in Function Block

This chapter describes the PID (Proportional plus Integral plus Derivative) built-in function block, which is used for closed-loop process control. The PID function compares feedback from a process variable (PV) with a desired process set point (SP) and updates a control variable (CV) based on the error.

The PID function uses PID loop gains and other parameters stored in a 40-word reference array of 16-bit integer words to solve the PID algorithm at the desired time interval.

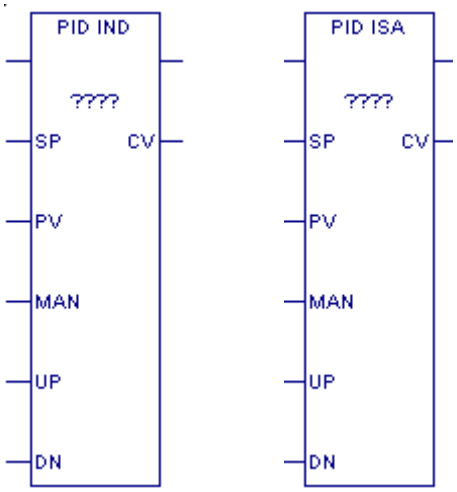


Ladder Diagram ***Function Block Diagram***

This chapter presents the following topics:

- Operands of the PID Function
- Reference Array for the PID Function
- PID Algorithm Selection and Gain Calculations
- Determining the Process Characteristics
- Setting and Tuning Loop Gains
- Example

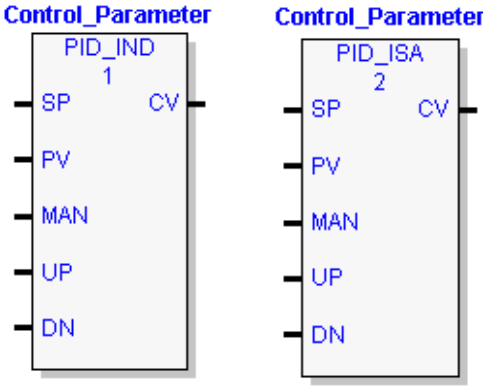
Operands of the PID Function



Operands for LD Version of PID Function Block

Parameter	*Description*	*Allowed Types*	*Allowed Operands*	*Optional*
Address (????)	Location of the PID reference array, which contains user-configurable and internal parameters, described on page 11-4. Uses 40 words that cannot be shared.	WORD	R, L, P, W and symbolic	No
SP	The control loop or process Set Point. Set using Process Variable counts, the PID function adjusts the output Control Variable so that the Process Variable matches the Set Point (zero error).	INT, BOOL array of length 16 or more	All except S, SA, SB, and SC	No
PV	Process Variable input from the process being controlled. Often a %AI input.	INT, BOOL array of length 16 or more	All except S, SA, SB, and SC, and constant	No
MAN	When energized to 1 (through a contact), the PID function block is in *manual* mode. If this input is 0, the PID function block is in *automatic* mode.	NA	Flow	No
UP	If energized along with MAN, increases the Control Variable by 1 CV count per solution of the PID function.	NA	Flow	No
DN	If energized along with MAN, decreases the Control Variable by 1 CV count per solution of the PID function..	NA	Flow	No
CV	The Control Variable output to the process. Often a %AQ output.	INT, BOOL array of length 16 or more	All except %S and constant	No

Operands for FBD Version of PID Function Block



Parameter	*Description*	*Allowed Types*	*Allowed Operands*	*Optional*
Parameter above the instruction	PID reference array, which contains user-configurable and internal parameters, described on page 11-4. Uses 40 words that cannot be shared.	WORD	R, L, P, W and symbolic	No
Function block solve order – FBD version	Calculated by the FBD editor.	NA	NA	No
SP	The control loop or process Set Point. Set using Process Variable counts, the PID function adjusts the output Control Variable so that the Process Variable matches the Set Point (zero error).	INT, BOOL array of length 16 or more	All except S, SA, SB, and SC	No
PV	Process Variable input from the process being controlled. Often a %AI input.	INT, BOOL array of length 16 or more	All except S, SA, SB, SC and constant	No
MAN	When energized to 1 (through a contact), the PID function block is in *manual* mode. If this input is 0, the PID block is in *automatic* mode.	BOOL	All	No
UP	If energized along with MAN, increases the Control Variable by 1 CV count per solution of the PID function block.	BOOL	All	No
DN	If energized along with MAN, decreases the Control Variable by 1 CV count per solution of the PID function block.	BOOL	All	No
CV	The Control Variable output to the process. Often a %AQ output.	INT, BOOL array of length 16 or more	All except %S and constant	No

Reference Array for the PID Function

This parameter block for the PID function occupies 40 words of memory, located at the starting Address specified in the PID function block operands. Some of the words are configurable. Other words are used by the CPU for internal PID storage and are normally not changed.

Every PID function call must use a different 40-word memory area, even if all the configurable parameters are the same.

The configurable words of the reference array must be specified before executing the PID function. Zeros can be used for most default values. Once suitable PID values have been chosen, they can be defined as constants in BLKMOV functions so the program can set and change them as needed.

The LD version of the PID function does not pass power flow if there is an error in the configurable parameters. The function can be monitored using a temporary coil while modifying data.

Note: If you set the Override bit (bit 0) of the Control Word (word 15 of the reference array) to 1, the next four bits of the control word must be set to control the PID function block input contacts, and the Internal SP (word 16) and Internal PV (word 18) must be set because you have taken control of the PID function block away from the ladder logic.

Scaling Input and Outputs

All parameters of the PID function are 16 bit integer words for compatibility with 16 bit analog process variables. Some parameters must be defined in either PV counts or units or in CV counts or units.

The SP input must be scaled over the same range as the PV, because the PID function calculates error by subtracting these two inputs.

The process PV and control CV counts do not have to use the same scaling. Either may be -32000 or 0 to 32000 to match analog scaling, or from 0 to 10000 to display variables as 0.00% to 100.00%. If the process PV and control CV do not use the same scaling, scale factors are included in the PID gains.

Reference Array Parameters

Note: Machine Edition software allows you to modify the configurable parameters for a PID instruction in real time in online programmer mode. To customize PID parameters, right click the PID function and select Tuning.

Words	*Parameter/Description*	*Low Bit Units*	*Range*
1 (Address + 0)	**Loop Number** Optional number of the PID block. It provides a common identification in the CPU with the loop number defined by an operator interface device.	Integer	0 to 255 (for user display only)
2 (Address + 1)	**Algorithm** 1 = ISA algorithm 2 = independent algorithm	-	Set by the CPU
3 (Address + 2)	**Sample Period** The shortest time, in 10 ms. increments, between solutions of the PID algorithm. For example, use a 10 for a 100 ms. sample period.	10 ms.	0 (every sweep) to 65535 (10.9 Min) At least 10ms.
4,5 (Address + 3, Address + 4)	**Dead Band +** **Dead Band –** Integral values defining the upper (+) and lower (-) Dead Band limits. If no Dead Band is required, these values must be 0. If the PID Error (SP - PV) or (PV - SP) is above the (-) value and below the (+) value, the PID calculations are solved with an Error of 0. If non-zero, the (+) value must greater than 0 and the (-) value less than 0 or the PID block will not function. Leave these at 0 until the PID loop gains are set up or tuned. A Dead Band might be added to avoid small CV output changes due to variations in error.	PV Counts	Dead Band +: 0 to 32767 (never negative) Dead Band -: -32768 to 0 (never positive)
6 (Address + 5)	**PID_IND: Proportional Gain (Kp)** **PID_ISA: Controller gain (Kc = Kp)** PID_IND: Change in the Control Variable in CV Counts for a 100 PV Count change in the Error term. Entered as an integer representing a fixed-point decimal ratio with two decimal places. Displayed as a ratio of percentages with two decimal places. For example, a Kp entered as 450 is displayed as 4.50 and results in a Kp * Error / 100 or 450 * Error / 100 contribution to the PID Output. PID_ISA: Same as PID_IND. Kp is generally the first gain set when adjusting a PID loop.	0.01 CV%/PV% %CV / %PV	0 to 327.67%
7 (Address + 6)	**PID_IND: Derivative Gain (Kd)** **PID_ISA: Derivative Time (Td = Kd)** PID_IND: Change in the Control Variable in CV Counts if the Error or PV changes 1 PV Count every 10 ms. Entered as an integer representing a fixed-point decimal time in seconds with two decimal places. The least significant digit represents 0.01 second (10 ms.) units. Displayed as seconds with two decimal places. For example, Kd entered as 120 is displayed as 1.20 Sec and results in a Kd * Δ Error / delta time or 120 * 4 / 3 contribution to the PID Output if Error changing by 4 PV Counts every 30ms. Kd can be used to speed up a slow loop response, but is very sensitive to PV input noise. This noise sensitivity can be reduced by using the derivative filter, which is enabled by setting bit 5 of the Config Word (see page 11-7.) PID_ISA: The ISA derivative time in seconds, Td, is entered and displayed in the same way as Kd. Total derivative contribution to PID Output is Kc * Td * Δ Error / dt.	0.01 seconds	0 to 327.67 sec

Words	Parameter/Description	Low Bit Units	Range
8 (Address + 7)	**PID_IND: Integral Rate (Ki)** **PID_ISA: Integral Rate (1/Ti = Ki)** PID_IND: Rate of change in the Control Variable in CV Counts per second when the Error is a constant 1 PV Count. Entered as an integer representing a fixed-point decimal rate with three decimal places. The least significant digit represents 0.001 counts per second, or 1 count per 0.001 second. Displayed as Repeats/Sec with three decimal places. For example, Ki entered as 1400 is displayed as 1.400 Repeats/Sec and results in a Ki * Error * *dt* or 1400 * 20 * 50/1000 = 1,400 contribution to PID Output for an Error of 20 PV Counts and a 50 ms. CPU sweep time (Sample Period of 0). PID_ISA: The ISA Integral Time in seconds, Ti, must be inverted and entered, as integral rate, as described for PID_IND. Total integral contribution to PID Output is Kc * Ki * Error * *dt*. Ki is usually the second gain set after Kp.	Repeats/0.001 Sec.	0 to 32.767 repeats/sec
9 (Address + 8)	**CV Bias/Output Offset** Number of CV Counts added to the PID Output before the rate and amplitude clamps. It can be used to set non-zero CV values when only Kp Proportional gains are used, or for feed-forward control of this PID loop output from another control loop.	CV Counts	-32768 to 32767 (add to PID output)
10, 11 (Address + 9. Address + 10)	**CV Upper Clamp** **CV Lower Clamp** Number of CV Counts that define the highest and lowest value that CV is allowed to take. These values are required. The Upper Clamp must have a more positive value than the Lower Clamp, or the PID block will not work. These are usually used to define limits based on physical limits for a CV output. They are also used to scale the Bar Graph display for CV. The PID block has anti-reset-windup, controlled by bit 4 of the Config Word, to modify the integral term value when a CV clamp is reached.	CV Counts	-32768 to 32767 (Word 10 must be greater than word 11.)
12 (Address + 11)	**Minimum Slew Time** Minimum number of seconds for the CV output to move from 0 to full travel of 100% or 32000 CV Counts. It is an inverse rate limit on how fast the CV output can change. If positive, CV cannot change more than 32000 CV Counts times the solution time interval (seconds) divided by Minimum Slew Time. For example, if the Sample Period is 2.5 seconds and the Minimum Slew Time is 500 seconds, CV cannot change more than 32000 * 2.5 / 500 or 160 CV Counts per PID solution. The integral term value is adjusted if the CV rate limit is exceeded. When Minimum Slew Time is 0, there is no CV rate limit. Set Minimum Slew Time to 0 while tuning or adjusting PID loop gains.	Seconds / Full Travel	0 (none) to 32000 sec to move full CV travel

Words	Parameter/Description	Low Bit Units	Range
13 (Address + 12)	**Config Word** The low 6 bits of this word are used to modify default PID settings. The other bits should be set to 0. **Bit 0:** Error Term Mode. When this bit is 0, the error term is SP - PV. When this bit is 1, the error term is PV - SP. Setting this bit to 1 modifies the standard PID Error Term from the normal (SP – PV) to (PV – SP), reversing the sign of the feedback term. This mode is used for reverse acting controls where the CV must go down when the PV goes up. **Bit 1:** Output Polarity. When this bit is 0, the CV output is the output of the PID calculation. When it is set to 1, the CV output is the negated output of the PID calculation. Setting this bit to 1 inverts the Output Polarity so that CV is the negative of the PID output rather than the normal positive value. **Bit 2:** Derivative action on PV. When this bit is 0, the derivative action is applied to the error term. When it is set to 1, the derivative action is applied to PV only. **Bit 3:** Deadband action. When the Deadband action bit is 0, the actual error value is used for the PID calculation. When the Deadband action bit is 1, deadband action is chosen. If the error value is within the deadband limits, the error used for the PID calculation is forced to be zero. If, however, the error value is outside the deadband limits, the magnitude of the error used for the PID calculation is reduced by the deadband limit (|error| = |error – deadband limit|). **Bit 4:** Anti-reset windup action. When this bit is 0, the anti-reset-windup action uses a reset (integral term) back-calculation. When the output is clamped, the accumulated integral term is replaced with whatever value is necessary to produce the clamped output exactly. When the bit is 1, the accumulated integral term is replaced with the value of the integral term at the start of the calculation. In this way, the pre-clamp integral value is retained as long as the output is clamped. This option is not recommended for new applications. See “CV Amplitude and Rate Limits” on page 11-14. **Bit 5:** Enable derivative filtering. When this bit is set to 0, no filtering is applied to the derivative term. When set to 1, a first order filter is applied. This will limit the effects of higher frequency process disturbances, such as measurement noise, on the derivative term. **Setting Config Word:** Set Config Word to 0 for default operation. Add 1 (16#0001) to set bit 0, add 2 (16#0002) to set bit 1, add 4 (16#0004) to set bit 2, add 8 (16#0008) to set bit 3, add 16 (16#0010) to set bit 4, and add 32 (16#0020) to set bit 5. For example, to set bits 0, 3 and 5 only, set Config Word to 1 + 8 + 32 = 41 (16#0029). Some users will find the Config Word value easier to interpret in hexadecimal (16#) format.	Low 6 bits used	Boolean
14 (Address + 13)	**Manual Command** Set to the current CV output while the PID block is in Automatic mode. When the block is switched to Manual mode, this value is used to set the CV output and the internal value of the integral term within the Upper and Lower Clamp and Slew Time limits.	CV Counts	Tracks CV in Auto or sets CV in Manual

<table>
<tr><th>Words</th><th>Parameter/Description</th><th>Low Bit Units</th><th>Range</th></tr>
<tr><td>15
(Address + 14)</td><td>Control Word
If the Override bit (bit 0) is set to 1, the Control Word and the internal SP, PV and CV parameters must be used for remote operation of the PID block (see below). This allows a remote operator interface device, such as a computer, to take control away from the PLC program.
Caution: If you do not want to allow remote operation of the PID block, make sure the Control Word is set to 0. If the low bit is 0, the next 4 bits can be read to track the status of the PID input contacts as long as the PID Enable contact has power.
Control Word is a discrete data structure with the first five bit positions defined in the following format:
<table>
<tr><th>Bit:</th><th>Word Value:</th><th>Function:</th><th>Status or External Action if Override bit is set to 1:</th></tr>
<tr><td>0</td><td>1</td><td>Override</td><td>If 0, monitor block contacts below. If 1, set them externally.</td></tr>
<tr><td>1</td><td>2</td><td>Manual /Auto</td><td>If 1, block is in Manual mode. If other numbers, it is in Automatic mode.</td></tr>
<tr><td>2</td><td>4</td><td>Enable</td><td>Should normally be 1. Otherwise block is never called.</td></tr>
<tr><td>3</td><td>8</td><td>UP /Raise</td><td>If 1 and Manual (Bit 1) is 1, CV is incremented every solution.</td></tr>
<tr><td>4</td><td>16</td><td>DN /Lower</td><td>If 1 and Manual (Bit 1) is 1, CV is decremented every solution.</td></tr>
</table></td><td>Maintained by the CPU, unless bit 0 (Override) is set to 1.</td><td>Boolean</td></tr>
<tr><td>16
(Address + 15)</td><td>Internal SP
Tracks the SP input. If Override = 1, must be set externally to solve the PID algorithm using an alternate SP value. The original SP value is maintained until overwritten.</td><td>Set and maintained by the CPU, unless bit 1 0 (Override) of Control Word is set to 1.</td><td>Non-configurable, unless bit 1 0 (Override) of Control Word is set to 1.</td></tr>
<tr><td>17
(Address + 16)</td><td>Internal CV
Tracks CV output.</td><td>Set and maintained by the CPU.</td><td>Non-configurable.</td></tr>
<tr><td>18
(Address + 17)</td><td>Internal PV
Tracks PV input. Must be set externally if Override bit is set to 1.</td><td>Set and maintained by the CPU, unless bit 1 0 (Override) of Control Word is set to 1.</td><td>Non-configurable, unless bit 1 0 (Override) of Control Word is set to 1.</td></tr>
<tr><td>19
(Address + 18)</td><td>Output
A Signed word value representing the output of the function block before the optional inversion. If the output polarity bit in the Config Word is set to 0, this value equals the CV output. If the output polarity bit is set to 1, this value equals the negative of the CV output.</td><td rowspan="4">Set and maintained by the CPU.</td><td rowspan="4">Non-configurable.</td></tr>
<tr><td>20
(Address + 19)</td><td>Derivative Term Storage
Used internally for storage of intermediate values. Do not write to this location.</td></tr>
<tr><td>21, 22
(Address + 20. Address + 21)</td><td>Integral Term Storage
Used internally for storage of intermediate values. Do not write to these locations.</td></tr>
<tr><td>23
(Address +22)</td><td>Slew Term Storage
Used internally for storage of intermediate values. Do not write to this location.</td></tr>
<tr><td>24 – 26
(Address + 23 – Address + 25)</td><td>Previous Solution Time
Internal storage of time of last PID solution. Normally do not write to these locations. Some special circumstances may justify writing to these locations.
Note: If you call the PID block in Automatic mode after a long delay, you might want to use SVC_REQ #16 or SVC_REQ #51 to load the current CPU elapsed time clock into Word 24 to update the last PID solution time to avoid a step change of the integral term.</td><td>Set and maintained by the CPU.</td><td>Non-configurable.</td></tr>
</table>

Words	Parameter/Description	Low Bit Units	Range
27 (Address + 26)	**Integral Remainder Storage** Holds remainder from integral term scaling.	Set and maintained by the CPU.	Non-configurable.
28, 29 (Address + 27,Address + 28)	**SP, PV Lower Range** **SP, PV Upper Range** Optional integer values in PV Counts that define high and low display values for SP and PV. (Word 29 must be greater than word 28.)	PV Counts	-32768 to 32767
30 (Address + 29)	**Reserved** Word 30 is reserved. Do not use this location.	N/A	Non-configurable.
31, 32 (Address + 30, Address + 31)	**Previous Derivative Term Storage** Used in calculations for derivative filter. Do not write to these locations.	Set and maintained by the CPU.	Non-configurable.
33 – 40 (Address + 32 – Address + 39)	**Reserved** Words 32-39 are reserved. Do not use these references.	N/A	Non-configurable

Operation of the PID Function

Automatic Operation

The PID function can be called as often as every 10 milliseconds by providing power flow to the Enable input and no power flow to the Manual input. The function block compares the current CPU time with the last PID solution time stored in the reference array. If the interval between the two times is equal to or greater than the Sample Period (word 3 of the reference array) and also equal to or greater than 10 milliseconds, the PID algorithm is solved using this time interval. Both the last solution time and CV output are updated. In Automatic mode, the output CV is placed in the Manual Command parameter (word 14 of the reference array).

Note: If you call the PID block in Auto mode after a long delay, you may want to use SVC_REQ 16 or SVC_REQ 51 to load the current CPU time into the stored previous solution time (word 24 of the reference array, described on page 11-8). This will update the last PID solution time and avoid a large step change of the integral term.

Manual Operation

The PID function block is placed in Manual mode by providing power flow to both the Enable and Manual input contacts. The output CV is set from the Manual Command parameter. If either the UP or DN inputs have power flow, the Manual Command word is incremented (UP) or decremented (DN) by one CV count every PID solution. For faster manual changes of the output CV, it is also possible to add or subtract any CV count value directly to/from the Manual Command word (word 14 of the reference array).

The PID function block uses the CV Upper Clamp and CV Lower Clamp parameters to limit the CV output. If a positive Minimum Slew Time (word 12 of the reference array) is defined, it is used to limit the rate of change of the CV output. If either CV Clamp or the rate of change limit is exceeded, the value of the integral (reset) term is adjusted so that CV is at the limit. The anti-reset-windup feature assures that when the error term tries to drive CV above (or below) the clamps for a long period of time, the CV output will move off the clamp immediately when the error term changes sufficiently.

This operation, with the Manual Command tracking CV in Automatic mode and setting CV in Manual mode, provides a bump-less transfer between Automatic and Manual modes. The CV Upper and Lower Clamps and the Minimum Slew Time always apply to the CV output in Manual mode and the integral term is always updated. This assures that when a user rapidly changes the Manual Command value in Manual mode, the CV output cannot change any faster than the slew rate limit set by the Minimum Slew Time, and the CV cannot go above the CV Upper Clamp limit or below the CV Lower Clamp limit.

Time Interval for the PID Function

The start time of each PLC sweep is used as the current time when calculating the time interval between solutions of the PID function. The times and time intervals have a resolution of 100 microseconds. When an application uses multiple PID functions, all of them use the same time value.

The PID algorithm is solved when the current time is equal to or greater than the time of the last PID solution plus the Sample Period or 10 milliseconds; whichever is larger. If the Sample Period is set for execution on every sweep (value = 0), the PID function is restricted to a minimum of 10 milliseconds between solutions. ***If the sweep time is less than 10 milliseconds, the PID function waits until enough sweeps have occurred to accumulate an elapsed time of at least 10 milliseconds.*** For example, if the sweep time is 9 milliseconds, the PID function executes every other sweep, and the time interval between solutions is 18 milliseconds. If a specific PID function is executed more than once per sweep (by referencing the same reference array location in multiple PID function blocks), the algorithm is solved only on the first call.

The longest possible interval between executions is 65,535 times 10 milliseconds, or 10 minutes, 55.35 seconds.

PID Algorithm Selection (PIDISA or PIDIND) and Gain Calculations

The PID function supports both the Independent Term (PID_IND) and ISA standard (PID_ISA) forms of the PID algorithm. The Independent Term form takes its name from the fact that the coefficients for the proportional, integral and derivative terms act independently. The ISA algorithm is named for the Instrument Society of America (now the International Society for Measurement and Control), which standardized and promoted it.

The two algorithms differ in how words 6 through 8 of the reference array are used and in how the PID output (CV) is calculated.

The Independent term PID (PID_IND) algorithm calculates the output as:

PID Output = Kp * Error + Ki * Error * *dt* + Kd * Derivative + CV Bias

where Kp is the proportional gain, Ki is the integral rate, Kd is the derivative time, and *dt* is the time interval since the last solution.

The ISA (PID_ISA) algorithm has different coefficients for the terms:

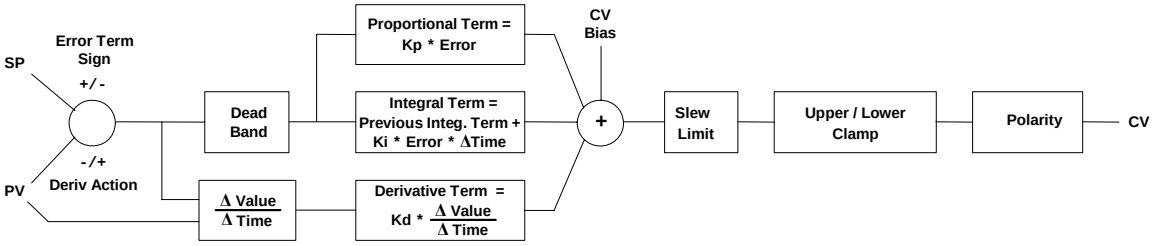
PID Output = Kc * (Error + Error * *dt*/Ti + Td * Derivative) + CV Bias

where Kc is the controller gain, Ti is the Integral time and Td is the Derivative time. The advantage of PID_ISA is that adjusting Kc changes the contribution for the integral and derivative terms as well as the proportional term, which can simplify loop tuning.

If you have the PID_ISA Kc, Ti and Td values, use the following equations to convert them to use as PID_IND parameters:

Kp = Kc, Ki = Kc/Ti, and Kd = Kc * Td

The following diagram shows how the PID_IND algorithm works:



The ISA Algorithm (PID_ISA) is similar except that its Kc gain coefficient is applied after the three terms are summed, so that the integral gain is Kc / Ti and the derivative gain is Kc * Td.

Bits 0, 1 and 2 in the Config Word set the Error sign, Output Polarity and Derivative Action, respectively.

Error Term

Both PID algorithms calculate the Error term as

Error = (SP – PV),

which can be changed to Reverse Acting mode:

Error = (PV – SP)

by setting the Error Term mode (bit 0) in the Config Word (word 13 of the reference array) to 1.

Reverse Acting mode is used if you want the CV output to move in the opposite direction from PV input changes (CV down for PV up) instead of the normal CV up for PV up.

Bit 0 of Config Word (Word 13)	*Error Term Mode*	*Error Term*
0	Normal	SP – PV
1	Reverse Acting	PV – SP

Derivative Term

The Derivative Term is Kd (word 7 of the reference array) multiplied by the time rate of change of the Error term in the interval since the last PID solution.

Derivative = Kd * ΔError / dt = Kd * (Error – previous Error) / dt

where

dt = Current PLC time – PLC time at previous PID solution.

Two bits in the Config Word (word 13 of the reference array) affect the calculation of ΔError. There are four cases to consider.

In Normal mode, the change in the error term is:

ΔError = (Error – previous Error) = (SP – PV) – (previous SP – previous PV)
= (previous PV – PV) – (previous SP – SP) = – ΔPV + ΔSP = ΔSP – ΔPV

where

ΔPV = (PV – previous PV), and ΔSP = (SP – previous SP).

However, in Reverse-Acting mode, the current error term is (PV – SP), and the sign of the change in the error term is reversed:

ΔError = (Error – previous Error) = (PV – SP) – (previous PV – previous SP)
= (PV – previous PV) – (SP – previous SP) = ΔPV – ΔSP.

The change in the error term depends on changes in both SP and PV. If SP is constant,

ΔSP = 0,

and SP has no effect on the derivative term. When SP changes, however, it can cause large transient swings in the derivative term and hence the output. Loop stability may be improved by eliminating the effect of SP changes on the derivative term. To calculate the Derivative based only on the change in PV, set bit 2 of the Config Word to 1. This modifies the equations above by assuming SP is constant (ΔSP = 0).

For bit 2 set in normal mode (bit 0 = 0): ΔError = – ΔPV,

For bit 2 set in Reverse-Acting mode (bit 0 = 1): ΔError = ΔPV.

CV Bias Term

The CV Bias term (word 9 in the reference array) is an additive term separate from the PID inputs. It may be useful if you are using only Proportional gain (Kp) and you want the CV to be a non-zero value when the PV equals the SP and the Error is 0. In this case, set the CV Bias to the desired CV when the PV is at the SP. CV Bias can also be used for feed forward control where another PID loop or control algorithm is used to adjust the CV output of this PID loop.

If a non-zero Integral rate is used, the CV Bias will normally be 0 as the integral term acts as an automatic bias or "reset." Just start up in Manual mode and use the Manual Command word (word 14 of the reference array) to set the desired CV, and then switch to Automatic mode. This will immediately calculate the required value for the integral term.

CV Amplitude and Rate Limits

The PID block does not send the calculated Output directly to CV. Both PID algorithms can impose amplitude and rate of change limits on the output Control Variable. If the Minimum Slew Time (word 12 of the reference array) is non-zero, the rate of change (slew rate) limit is determined by dividing the maximum CV value (32,000) by the Minimum Slew Time. For example, if the Minimum Slew Time is 100 seconds, the rate limit will be 320 CV counts per second. If the solution interval was 50 milliseconds, the new CV output cannot change more than 320*50/1000 or 16 CV counts from the previous CV output.

The CV output is then compared to the CV Upper Clamp and CV Lower Clamp values (words 10 and 11 of the reference array). If CV is outside either limit, the CV output is clamped to the appropriate limit value. When the CV output is modified to impose either slew rate or amplitude limits (or both), the stored integral term would normally accumulate a large value over time. This phenomenon is known as "reset windup." Reset windup introduces errors in CV after the PID output no longer needs to be limited. For example, windup would prevent the CV output from moving off a clamp value immediately.

There are two optional methods for preventing reset windup. If the Anti-reset-windup Action bit (bit 4) of Config Word (word 13 of the reference array) is zero (the default), the integral term is adjusted at each PID solution to match the error input and limited CV output exactly. When PV changes while CV is clamped, or when CV is both rate and amplitude limited in a particular PID solution, this option assures that a smooth transition will always occur after CV is no longer limited.

If the Anti-reset-windup Action bit of Config Word is set, then the integral term stored on the previous PID solution is simply retained as long as CV is limited. This option was added to assure compatibility with existing PID applications when the default action described above was introduced. This option is not recommended for new applications.

Finally, the PID block checks the Output Polarity (bit 2 of the Config Word) and changes the sign of the output if the bit is 1.

CV = – Clamped PID Output if Output Polarity bit set, or

Clamped PID Output if Output Polarity bit cleared.

If the block is in Automatic mode, the final CV is placed in the Manual Command (word 14 of the reference array). If the block is in Manual mode, the PID equation is skipped because CV is set by the Manual Command, but the slew rate and amplitude limits are still checked. This assures that the Manual Command cannot change the output above the CV Upper Clamp or below the CV Lower Clamp, and the output cannot change faster than allowed by the Minimum Slew Time.

Sample Period and PID Function Block Scheduling

The PID function block is a digital implementation of an analog control function, so the *dt* sample time in the PID Output equation is not the infinitesimally small sample time available with analog controls. The majority of processes being controlled can be approximated as a gain with a first or second order lag and (possibly) a pure time delay. The PID function block sets a CV output to the process and uses the process feedback PV to determine an Error to adjust the next CV output. A key process parameter is the total time constant, which is how fast the process can change PV when the CV is changed. As discussed in "Determining the Process Characteristics" on page 11-16, the total time constant, T_p+T_c, for a first order system is the time required for PV to reach 63% of its final value when CV is stepped. The PID function block will not be able to control a process unless its Sample Period is well under half the total time constant. Larger Sample Periods will make it unstable.

The Sample Period should be no bigger than the total time constant divided by 10 (or down to 5 worst case). For example, if PV seems to reach about 2/3 of its final value in 2 seconds, the Sample Period should be less than 0.2 seconds, or 0.4 seconds worst case. On the other hand, the Sample Period should not be too small, such as less than the total time constant divided by 1000, or the Ki * Error * *dt* term for the PID integral term will round down to 0. For example, a very slow process that takes 10 hours or 36,000 seconds to reach the 63% level should have a Sample Period of 40 seconds or longer.

Variations of the time interval between PID function solutions can have short-term effects on the CV output. For example, if a step change to PV caused by measurement noise occurs between solutions, the value of the derivative term will be inversely proportional to the time interval. The performance of PID loops that are tuned for quick response may be improved when the solution interval is held constant by configuring the PLC CPU for constant sweep mode. Depending on the CPU model and the application, constant sweep times of 10 milliseconds, integer multiples of 10 milliseconds, or exact divisors of 10 milliseconds (1, 2 or 5 milliseconds) will be possible. The Sample Period can then be set for a suitable multiple of 10 milliseconds.

If many PID loops are used, allowing the application to solve all the loops on the same sweep may lead to wide variations in CPU sweep time. If the loops have a common Sample Period that is at least equal to the number of PID loops times the sweep time, a simple solution is to sequence one or more 1 bits through an array of zero bits and to use these bits to enable power flow to individual PID function blocks. The logic should assure that each PID function block is enabled no more often than its Sample Period.

Determining the Process Characteristics

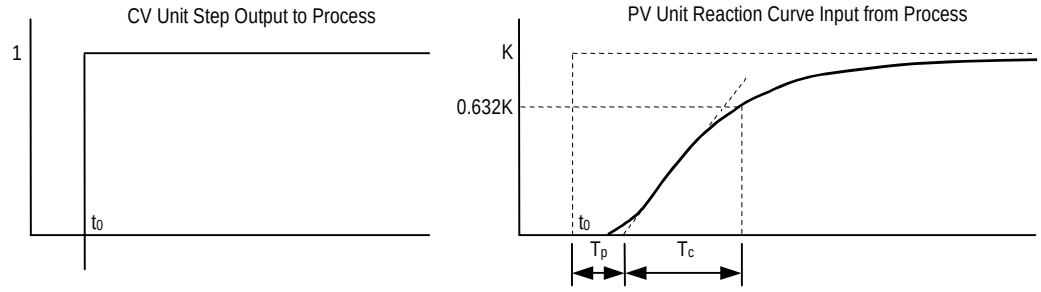
The PID loop gains, Kp, Ki and Kd, are determined by the characteristics of the process being controlled. Two key questions when setting up a PID loop are:

1. How big is the change in PV when CV is changed by a fixed amount, or what is the open loop gain of the process?
2. How fast does the system respond, or how quickly does PV change after the CV output is stepped?

Many processes can be approximated by a process gain, first or second order lag and a pure time delay. In the frequency domain, the transfer function for a first order lag system with a pure time delay is:

$$\frac{PV(s)}{CV(s)} = G(s) = Ke^{-T_p/(1+T_c s)}$$

Plotting the response to a step input at time t_0 in the time domain provides an open-loop unit reaction curve:



The following process model parameters can be determined from the PV unit reaction curve:

K	Process open loop gain = final change in PV/change in CV at time t_0 (Note no subscript on K)
T_p	Process or pipeline time delay or dead time after t_0 before the process output PV starts moving
T_c	First order Process time constant, time required after T_p for PV to reach 63.2% of the final PV

Usually the quickest way to measure these parameters is by putting the PID function block in Manual mode, making a small step change in the CV output by changing the Manual Command (word 14 of the reference array), and then plotting the PV response over time. For slow processes this can be done manually, but for faster processes a chart recorder or computer graphic data-logging package will help. The CV step size should be large enough to cause an observable change in PV, but not so large that it disrupts the process being measured. A good step size may be from 2 to 10% of the difference between the CV Upper and CV Lower Clamp values.

Setting Tuning Loop Gains

Basic Iterative Tuning Approach

Because PID parameters are dependent on the process being controlled, there are no predetermined values that will work. However, a simple iterative process can be used to find acceptable values for Kp, Ki, and Kd gains.

1. Set all the reference array parameters to 0, then set the CV Upper and CV Lower Clamps to the highest and lowest CV expected. Set the Sample Period to a value within the range $T_c/10$ to $T_c/100$, where T_c is the estimated process time constant defined on page 11-16.
2. Put the PID function block in Manual mode and set the Manual Command (word 14 in the reference array) to different values to check if CV can be moved to Upper and Lower Clamp. Record the PV value at some CV point and load it into SP.
3. Set a small gain, such as 100 * Maximum CV/Maximum PV, into Kp and turn off Manual mode. Step SP by 2% to 10% of the Maximum PV range and observe PV response. Increase Kp if PV step response is too slow or reduce Kp if PV overshoots and oscillates without reaching a steady value.
4. Once a Kp is found, start increasing Ki to get overshooting that dampens out to a steady value in two to three cycles. This may require reducing Kp. Also try different SP step sizes and CV operating points.
5. After suitable Kp and Ki gains are found, try adding Kd to get quicker responses to input changes providing it doesn't cause oscillations. Kd is often not needed and will not work with noisy PV.
6. Check gains over different SP operating points and add Dead Band and Minimum Slew Time if needed. Some Reverse Acting processes may need setting of Config Word Error Term or Output Polarity bits.

Setting Loop Gains Using the Ziegler and Nichols Tuning Approach

This approach provides good response to system disturbances with gains producing an amplitude ratio of 1/4. The amplitude ratio is the ratio of the second peak over the first peak in the closed loop response.

1. Determine the three process model parameters, K, T_p and T_c for use in estimating initial PID loop gains.
2. Calculate the Reaction rate:
   $R = K/T_c$
3. For Proportional control only, calculate Kp as:
   $Kp = 1/(R * T_p) = T_c/(K * T_p)$

   For Proportional and Integral control, use:
   $Kp = 0.9/(R * T_p) = 0.9 * T_c/(K * T_p)$ $Ki = 0.3 * Kp/T_p$

   For Proportional, Integral and Derivative control, use:
   $Kp = G/(R * T_p)$ where G is from 1.2 to 2.0
   $Ki = 0.5 * Kp/T_p$
   $Kd = 0.5 * Kp * T_p$
4. Check that the Sample Period is in the range
   $(T_p + T_c)/10$ to $(T_p + T_c)/1000$

Ideal Tuning Method

The “Ideal Tuning" procedure provides the best response to SP changes that are delayed only by the T_p process delay or dead time.

1. Determine the three process model parameters, K, Tp and Tc for use in estimating initial PID loop gains.
2. Calculate Kp, Ki, and Kd as follows:

   $Kp = 2 * T_c/(3 * K * T_p)$
   $Ki = T_c$
   $Kd = Ki/4$ if Derivative term is used
3. Once initial gains are determined, convert them to integers.
4. Calculate the Process gain, K, as a change in input PV Counts divided by the resulting output step change in CV Counts. (Not in process PV or CV engineering units.) Specify all times in seconds.
5. Once Kp, Ki and Kd are determined, Kp and Kd are multiplied by 100 while Ki is multiplied by 1000. The resulting values are entered into the corresponding reference array word locations.

Example

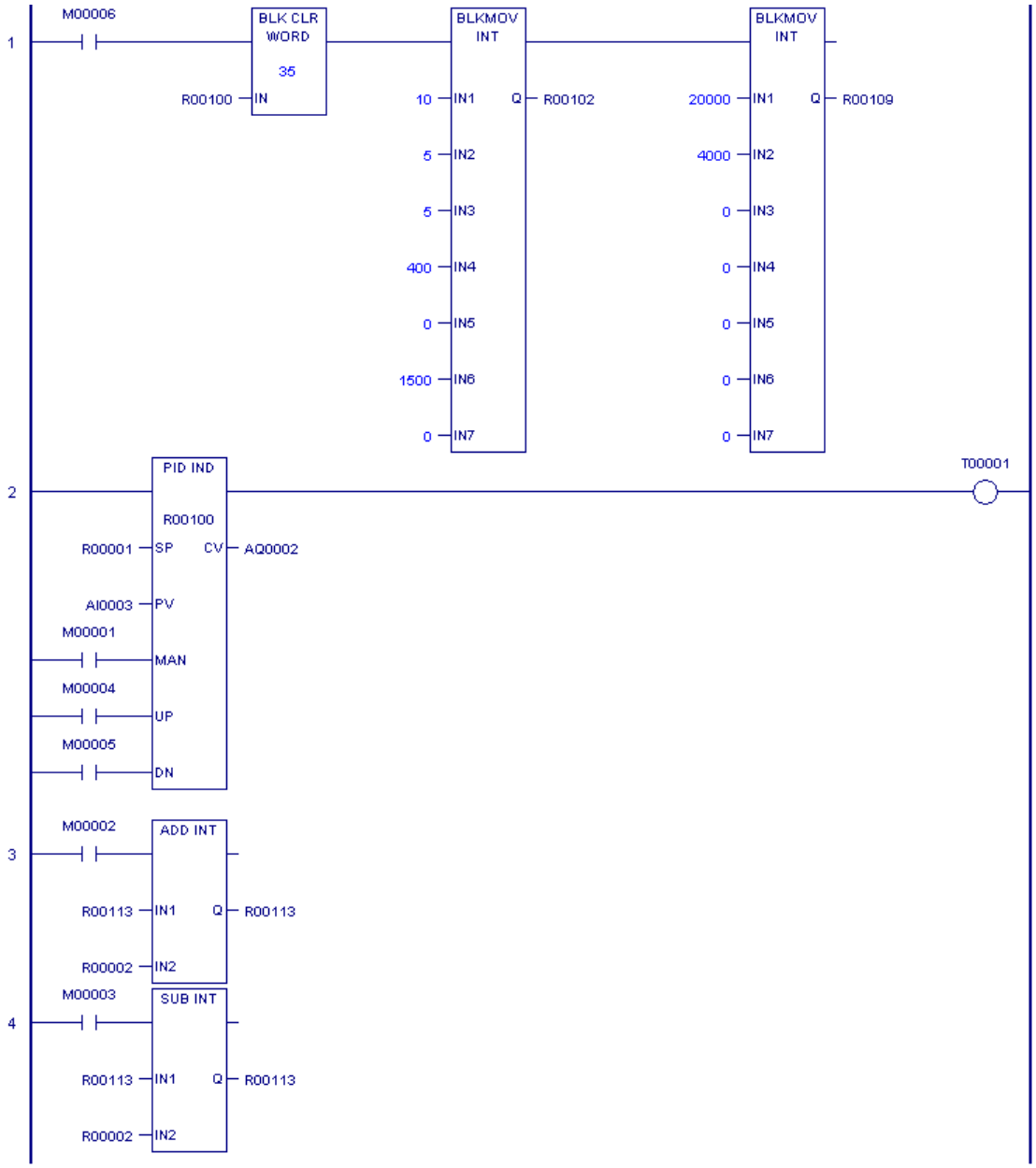
The following PID example has a sample period of 100 ms, a Kp gain of 4.00 and a Ki gain of 1.500. The set point is stored in %R0001, the control variable is output in %AQ0002, and the process variable is returned in %AI0003. CV Upper and CV Lower Clamps must be set, in this case to 20000 and 4000, and an optional small Dead Band of +5 and -5 is included. The 40-word reference array starts in %R0100. Normally, user parameters are set in the reference array, but %M0006 can be set to reinitialize the 14 words starting at %R0102 (word 3) from constants stored in logic (a useful technique).

The block can be switched to Manual mode with %M1 so that the Manual Command, %R113, can be adjusted. Bits %M4 or %M5 can be used to increase or decrease %R113 and the PID CV by 1 every 100 ms solution. For faster manual operation, bits %M2 and %M3 can be used to add or subtract the value in %R2 to/from %R113 every CPU sweep. The %T1 output is on when the PID is OK.

Reference Array Initialization using %M00006

For details on the contents of the reference array, refer to page 11-4.

Word	*Function*	*Address*	*Value*
3	Sample Period	%R102	10
4	+ Dead Band	%R103	5
5	- Dead Band	%R104	5
6	Kp	%R105	400
7	Kd	%R106	0
8	Ki	%R107	1500
9	CV Bias	%R108	0
10	CV Upper Clamp	%R109	2000
11	CV Lower Clamp	%R110	400
12	Minimum Slew Time	%R111	0
13	Config Word	%R112	0
14	Manual Command	%R113	0
15	Control Word	%R114	0
16	Internal SP	%R115	0


1
M00006
BLK CLR WORD
35
R00100
IN
BLKMOV INT
10
IN1
Q
R00102
5
IN2
5
IN3
400
IN4
0
IN5
1500
IN6
0
IN7
BLKMOV INT
20000
IN1
Q
R00109
4000
IN2
0
IN3
0
IN4
0
IN5
0
IN6
0
IN7
2
PID IND
R00100
R00001
SP
CV
AQ0002
AI0003
PV
M00001
MAN
M00004
UP
M00005
DN
T00001
3
M00002
ADD INT
R00113
IN1
Q
R00113
R00002
IN2
4
M00003
SUB INT
R00113
IN1
Q
R00113
R00002
IN2

Chapter

12

Structured Text Programming

The Structured Text (ST) programming language is an IEC 61131-3 textual programming language. It is convenient for those who have experience with high-level programming languages, such as C. Structured text also allows greater flexibility in writing algorithms and can be easily transferred between different types of controllers. Its compactness allows you to view a complex algorithm on one screen.

This chapter describes how structured text is implemented in PACSystems. For information on using the structured text editor in the programming software, refer to the online help.

The block types Block, Parameterized Block, and Function Block (UDFB) can be programmed in ST. The _MAIN program block can also be programmed in ST. For details on blocks, refer to chapter 6, “Program Organization.”

Language Overview

Statements

A structured text program consists of a series of statements, which are constructed from expressions and language keywords. A statement directs the PLC to perform a specified action. Statements provide variable assignments, conditional evaluations, iteration, and the ability to call built-in functions. PACSystems supports the statements described in “Statement Types” on page 12-4.

Expressions

Expressions calculate values from variables and constants. An expression can involve operators, variables, and constants. An example of a simple expression is ***(x + 5)***.

Composite expressions can be created by nesting simpler expressions, for example, ***(a + b) * (c + d) – 3.0 ** 4***.

Operators

The table below lists the operators that you can use within an expression. They are listed according to their evaluation precedence, which determines the sequence in which they are executed within the expression. The operator with the highest precedence is applied first, followed by the operator with the next highest precedence. Operators of equal precedence are evaluated left to right. Operators in the same group, for example + and -, have the same precedence.

Any address operators used in LD can be used on ST operands. Address operators have precedence over the ST language operators. Address operators include indirect addressing (for example, @Var1), array indexing (for example, Var1[3]), bit within word addressing (for example, Var1.X[3]), and structure fields (for example, Var1.field1).

Precedence	*Operator*	*Operand Types*	*Description*
Group 1 (Highest)	(…)		Parenthesized expression
Group 2	-	INT, DINT, REAL, LREAL	Negation
	NOT	BOOL, BYTE, WORD, DWORD	Boolean complement
Group 3	**,^	INT, DINT, UINT, REAL, LREAL[1]	Exponentiation[3, 5]
Group 4	*	INT, DINT, UINT, REAL, LREAL	Multiplication[3]
	/	INT, DINT, UINT, REAL, LREAL	Division[2, 3]
	MOD	INT, DINT, UINT	Modulus operation[2]
Group 5	+	INT, DINT, UINT, REAL, LREAL	Addition[3]
	-	INT, UINT, DINT, REAL, LREAL	Subtraction[3]
Group 6	<, >, <=, >=	INT, DINT, UINT, REAL, LREAL, BYTE, WORD, DWORD	Comparison
Group 7	=	ANY[4]	Equality
	<>, !=	ANY[4]	Inequality
Group 8	AND, &	BOOL, BYTE, WORD, DWORD	Boolean AND
Group 9	XOR	BOOL, BYTE, WORD, DWORD	Boolean exclusive OR
Group 10 (Lowest)	OR	BOOL, BYTE, WORD, DWORD	Boolean OR

1 The base must be type REALor LREAL. If the base is REAL, the power can be type INT, DINT, UINT, or REAL and the result is type REAL. If the base is type LREAL, the power must be LREAL and the result will be LREAL.

2 The CPU flags a divide by 0 error as an application fault.

3 Use of math operators can cause overflow or underflow. Overflow results are truncated.

4 Operators that can take operands of type ANY can be used with any of the supported elementary data types. The supported data types are: BOOL, INT, DINT, UINT, BYTE, WORD, DWORD, LREAL and REAL. STRING and TIME data types are not supported.

5 If either operand is positive or negative infinity, the result is undefined.

Some comparison and math operators have corresponding built-in functions. For instance the '+' operator is similar to the ADD_INT function. You can use either the language operator or the built-in function. The built-in function has the advantage of returning an ENO status. For information on built-in functions, see page 12-6.

Operand Types

Type casting is not supported. To convert a type, use one of the built-in conversion functions. Use of built-in functions is described in “Function Call” on page 12-6.

For untyped operators (+, *, …), the types of the operands must match.

Structured Text Syntax

The syntax of the ST implementation for PACSystems follows the IEC 61131-3 standard.

- Structured Text statements must end in a semi-colon (;).
- Structured Text variables must be declared in the variable list for the target.

These symbols have the following functions.

:= assigns an expression to a variable

; required to designate the end of a statement

[ ] used for array indexing where the array index is an integer. For example, this sets the third element of an array to the value j+10: ***intarray[3]: = j + 10***;

(* *) designates a comment. These comments can span multiple lines. For example,

```
(*This comment spans
        multiple lines.*)
```

// or ‘ designates a single line comment. For example,

```
c :=a+b; //This is a single line comment.
c :=a+b; ‘This is a single line comment.
```

Statement Types

The Structured Text statements, which specify the actual program execution, consist of the following types.

Statement Type	Description	Example
Assignment	Sets an object to a specified value.	A := 1; B := A; C := A + B;
CASE	Provides for the conditional execution of a set of statements.	CASE A OF 1,2 : C := 3; 3: C := 4; 4..5: C := 5; ELSE C := 0; END_CASE;
Function call	Calls a function for execution.	FbInst(IN1 := 1, OUT1 => A);
RETURN	Causes the program to return from a subroutine. The return statement provides an early exit from a block.	RETURN;
EXIT	Terminates iterations before the terminal condition becomes TRUE (1).	EXIT;

IF	Specifies that one or more statements be executed conditionally.	IF (A < B) THEN
		C := 4;
		ELSIF (A = B) THEN
		C:= 5;
		ELSE
		C := 6
		END_IF;

FOR	Executes a statement sequence repeatedly based on the value of a control symbol.	FOR I := 1 TO 100 BY 2 DO
		IF (Var1 – I) = 40 THEN
		Key := I;
		EXIT;
		END_IF;
		END_FOR;

WHILE	Indicates that a statement sequence be executed repeatedly until a Boolean expression evaluates to FALSE (0).	WHILE J <= 100 DO J := J + 2; END_WHILE;
REPEAT	Indicates that a statement sequence be executed repeatedly until a Boolean expression evaluates to TRUE (1).	REPEAT J := J + 2; UNTIL J >= 100 END_REPEAT;
ARG_PRESENT	Determines whether a parameter value was present when the function block instance of the parameter was invoked. For example, a parameter can be optional (pass by value).	ARG_PRES (IN :=In1, Q:>Out1, ENO:>Out2);
Empty Statement		;

Assignment Statement

The assignment statement replaces the value of a variable with the result of evaluating an expression (of the same data type).

Notes:

- Assignment statements can affect transition bits.
- Assignment statements take override bits into account.

Format

Variable := Expression;

Where:

Variable is a simple variable, array element, etc.

Expression is a single value, expression, or complex expression.

Examples

Boolean assignment statements:

```
VarBool1 := 1;
VarBool2 := (val <= 75);
```

Array element assignment:

```
Array_1[13] := (RealA /RealB)* PI;
```

Function Call

The structured text function call executes a predefined algorithm that performs a mathematical, bit string or other operation. The function call consists of the name of the function or block followed by required input or output parameters.

The structured text logic can call blocks or the PACSystems built-in functions listed in the table below. The call must be made in a single statement and cannot be part of a nested expression.

Calls to some functions, such as communications request (COMM_REQ), require a command block or parameter block. For these functions, an array is declared, initialized in logic, and then passed as a parameter to the function.

Built-in Functions Supported for ST Calls

Note: Only the functions listed in the following table are supported in the current PACSystems version. Other built-in functions are not supported.

Advanced Math	ASIN, ATAN, ACOS, COS, SIN, TAN, DEG_TO_RAD, RAD_TO_DEG LOG, LN, EXP, EXPT, SQRT_INT, SQRT_DINT, SQRT_REAL ABS_INT, ABS_DINT, ABS_REAL SCALE_DINT, SCALE_INT, SCALE_UINT	For more information, see "Advanced Math" in chapter 8.
Control	DO_IO, MASK_IO_INTR, SCAN_SET_IO, SUS_IO, SUS_IO_INTR, SVC_REQ, SWITCH_POS, F_TRIG, R_TRIG	For more information, see "Control Functions" in chapter 8.
Data Conversion	BCD4_TO_INT, BCD4_TO_UINT, BCD4_TO_REAL BCD8_TO_DINT, BCD8_TO_REAL DINT_TO_BCD8, DINT_TO_INT, DINT_TO_UINT, DINT_TO_REAL UINT_TO_BCD4, UINT_TO_BCD8, UINT_TO_INT INT_TO_BCD4, INT_TO_DINT, INT_TO_UINT REAL_TO_INT, REAL_TO_UINT, REAL_TO_DINT TRUNC_INT, TRUNC_DINT	For more information, see "Conversion Functions" in chapter 8.
	DINT_TO_DWORD DWORD_TO_DINT UINT_TO_DINT, UINT_TO_REAL, UINT_TO_WORD INT_TO_REAL, INT_TO_WORD WORD_TO_INT, WORD_TO_UINT	For more information, see page 12-7.
Data Move	COMM_REQ,	For more information, see "Data Move" functions in chapter 8.

Conversion Functions

Note: ENO is an optional BOOL output parameter. If ENO is used in a statement that uses the formal convention, the state of *outBool* is set to 1 (call was successful) or 0 (call failed).

Mnemonic	***Description***	***Example (Formal Convention)***
WORD_TO_INT	Converts the input data into the equivalent single-precision signed integer (INT) value, which it outputs to Q.	word_to_int(IN := inWord, Q => outInt, ENO => outBool);
WORD_TO_UINT	Converts the input data into the equivalent single-precision unsigned integer (UINT) value, which it outputs to Q.	word_to_uint(IN := inWord, Q => outUint, ENO => outBool);
DWORD_TO_DINT	Converts DWORD data into the equivalent signed double-precision integer (DINT) value and stores the result in Q.	dword_to_dint(IN := inDword, Q => outDint, ENO => outBool);
UINT_TO_ WORD	Converts an unsigned single-precision integer (UINT) operand IN to a 16-bit bit string (WORD) value and stores the result in the variable assigned to Q.	uint_to_word(IN := inUint, Q => outWord, ENO => outBool);
INT_TO_WORD	Converts a 16-bit signed integer (INT) operand IN to a 16-bit bit string (WORD) value and stores the result in the variable assigned to Q.	int_to_word(IN := inInt, Q => outWord, ENO => outBool);
DINT_TO_DWORD	Converts the input double-precision signed integer (DINT) data into the equivalent DWORD (32-bit bit string) value, which it outputs to Q.	dint_to_dword(IN := inDint, Q => outDword ENO => outBool);

Block Types Supported for ST Calls

An ST block can call blocks of type Block, Parameterized Block, or user defined Function Block (UDFB). For more information on block types, refer to chapter 6.

Formal Calls vs. Informal Calls

PACSystems supports formal and informal calls in ST.

Formal Calls	***Informal Calls***
Input parameter assignments use the ':=' notation while output assignments use the '=>' notation.	Input and output parameters are listed in parentheses.
Optional parameters can be omitted.	Parameters cannot be omitted.
Parameters can be in any order.	Parameters must be in the correct order as follows: Inputs Instance location (if required) Length parameter (if required) Outputs, starting with the last output parameter.
The ENO parameter is specified in a formal function or block call. All built-in functions and user-defined blocks have an optional ENO output parameter indicating the success of the function or block. Either ENO or Y0 can be used as this output parameter name.	The ENO parameter is not specified in an informal function or block call.

Format of Formal Function Call

FunctionName(IN1 := inparam1, IN2 := inparam2, OUT1 => outparam1, ENO => enoparam);

Format of Informal Function Call

FunctionName(inparam1, inparam2, outparam1);

Example

This code fragment shows the TAN function call.

TAN(AnyReal, Result);

Calls to Standard Function Blocks

Standard function blocks are instructions that have instance data in the form of a structure variable. (For more information on function blocks and their instance data, refer to "Functions and Function Blocks" in chapter 6.) Standard function blocks are called in the same way that a UDFB is called.

PACSystems controllers support three standard function blocks:

Pulse timer (TP)	Generates output pulses of a given duration
On-delay timer (TON)	Delays setting an output ON for a fixed period after an input is set ON.
Off-delay timer (TOF)	Delays setting an output OFF for a fixed period after an input goes OFF so that the output is held on for a given period longer than the input.

For details on the operation of TP, TON and TOF, refer to "Standard Timer Function Blocks" in chapter 8.

Format of Calls to Standard Timer Function Blocks

Notes: TOF, TON and TP have the same input and output parameters, except for the instance variable, which must be the same type as the instruction.

Writing or forcing values to the instance data elements IN, PT, Q, ET, ENO or TI may cause erratic operation of the timer function block.

Instance data can be a variable or a parameter of the current UDFB or parameterized block.

Formal Convention

myTOF_Instance_Data(IN := inBool, PT := inDINT, ET => outDINT, Q => outBool, ENO => outBoolSuccess);

myTON_Instance_Data(IN := inBool, PT := inDINT, ET => outDINT, Q => outBool, ENO => outBoolSuccess);

myTP_Instance_Data(IN := inBool, PT := inDINT, ET => outDINT, Q => outBool, ENO => outBoolSuccess);

Note: ENO is an optional BOOL output parameter. If ENO is used in a statement that uses the formal convention, the state of *outBoolSuccess* is set to 1 (call was successful) or 0 (call failed).

Informal Convention

myTOF_Instance_Data(inBool, inDINT, outDINT, outBool);

myTON_Instance_Data(inBool, inDINT, outDINT, outBool);

myTP_Instance_Data(inBool, inDINT, outDINT, outBool);

Note: When using the informal convention, the operands must be assigned in the order shown above (that is, IN, PT, ET, Q and ENO).

RETURN Statement

The return statement provides an early exit from a block. For example, in the following lines of code the third line will never execute. The variable **a** will have the value 4.

```
a := 4;

RETURN;

a := 5;
```

IF Statement

The IF construct offers conditional execution of a statement list. The condition is determined by result of a Boolean expression. The IF construct includes two optional parts, ELSE and ELSIF, that provide conditional execution of alternate statement list(s). One ELSE and any number of ELSIF sections are allowed per IF construct.

Format

```
IF  BooleanExpression1  THEN
    StatementList1;
[ELSIF  BooleanExpression2  THEN    (*Optional*)
    StatementList2;]
[ELSE                                    (*Optional*)
    StatementList3;]
END_IF;
```

Where:

BooleanExpression Any expression that resolves to a Boolean value.

StatementList Any set of structured text statements.

Note: Either ELSIF or ELSEIF can be used for the else if clause in an IF statement.

Operation

The following sequence of evaluation occurs if both optional parts are present:

- If BooleanExpression1 is TRUE (1), StatementList1 is executed. Program execution continues with the statement following the END_IF keyword.
- If BooleanExpression1 is FALSE (0) and BooleanExpression2 is TRUE (1), StatmentList2 is executed. Program execution continues with the statement following the END_IF keyword.
- If both Boolean expressions are FALSE (0), StatmentList3 is executed. Program execution continues with the statement following the END_IF keyword.

If an optional part is not present, program execution continues with the statement following the END_IF keyword.

Example

The following code fragment puts text into the variable Status, depending on the value of I/O point input value.

```
IF Input01 < 10.0 THEN
    Status := Low_Limit_Warning;
ELSIF Input02 > 90.0 THEN
    Status := Upper_Limit_Warning;
ELSE
    Status := Limits_OK;
END_IF;
```

CASE Statement

The CASE …. OF construct offers conditional execution of statement lists. It uses the value of an ST integer expression to determine whether to execute a statement list. The statement list to be executed can be selected from multiple statement lists, depending on the value of the associated integer expression.

Conditions can be expressed as a single value, a list of values, or a range of values. The single-value, list of values, or range forms can be used by themselves or in combination. The optional ELSE keyword can be used to execute a statement list when the associated value does not meet any of the specified conditions.

You can have a maximum of 1024 cases in a single CASE … OF construct. Additional cases can be handled by adding the ELSE keyword to the construct and specifying a nested CASE … OF construct or an IF … THEN construct after the ELSE.

The number of nested CASE … OF constructs and the number of levels are limited by the memory in your computer.

The number of constants and constant ranges in a single conditional statement is limited by the memory in your computer.

Format

```
CASE        Integer_Expression OF
    Int1:                           (*Single Value*)
    StatementList_1;
    Int2,Int3,Int4:             (*List of Values*)
    StatementList_2;
    Int5..Int6:                     (*Range of Values*)
    StatementList_3;
[ELSE                               (*Optional*)
    StatementList_Else;]
END_CASE;
```

Where:

Integer_Expression An ST expression that resolves to an integer (INT, DINT or UINT) value.

Int A constant integer value.

StatementList_1 … StatementList_n
Structured Text statements.

Operation

The Int values are compared to Integer_Expression. The statement list following the first Int value that matches Integer_Expression is executed. If the optional ELSE keyword is used and no Int value matches Integer_Expression, the statement list following ELSE is executed. Otherwise, no statement list is executed.

Requirements for Conditional Statements

- All constants must be of type INT, DINT or UINT.
- In range declarations, the beginning value must be less than the ending value (reading from left to right). For example, 10..3 and 5..5 are invalid.
- Overlapping values in different case conditions are not allowed. For example, 5..10 and 7 cannot be specified as conditions in the same CASE … OF construct.

Examples

The following code fragment assigns a value to the variable ColorVariable.

```
CASE ColorSelection OF
    0:      ColorVariable:= Red;
    1:      ColorVariable:= Yellow;
    2,3,4:  ColorVariable:= Green;
    5..9:   ColorVariable:= Blue;
ELSE        ColorVariable:= Violet;
END_CASE;
```

The following code fragment uses a nested CASE…OF…END_CASE construct.

```
CASE ColorSelection OF
    0:      ColorVariable:= Red;
    1:      ColorVariable:= Yellow;
    2,3,4:  ColorVariable:= Green;
    5..9:   ColorVariable:= Blue;
ELSE
    CASE ColorSelection OF
            10: ColorVariable:= Violet;
    ELSE    ColorVariable:= Black;
    END_CASE;
    ColorError: 1;
END_CASE;
```

FOR Statement

The FOR loop repeatedly executes a statement list contained within the FOR … DO … END_FOR construct. It is useful when the number of iterations can be predicted in advance, for example to initialize an array. The number of iterations is determined by the value of a control variable which is incremented (or decremented) from an initial value to a final value by the FOR statement.

By default, each iteration of the FOR statement changes the value of the control variable by 1. The optional BY keyword can be used to specify an increment or decrement of the control variable by specifying a (non-zero) positive or negative integer or an expression that resolves to an integer.

FOR loops can be nested to a maximum of ten levels.

Format

```
FOR Control_Variable := Start_Value TO End_Value [BY Step_Value] DO
    Statement list;
END_FOR;
```

Where:

Control_Variable	The control variable. Can be an INT, DINT or UINT variable or parameter.
Start_Value	The starting value of the control variable. Must be an expression, variable, or constant of the same data type as Int_Variable.
End_Value	The ending value of the control variable. Must be an expression, variable, or constant of the same data type as Int_Variable.
Step_Value	(Optional) The increment or decrement value for each iteration of the loop. Must be an expression, variable, or constant of the same data type as Int_Variable. If Step_Value is not specified, the control variable is incremented by 1.
Statement list	Any list of Structured Text statements.

Operation

The values of Start_Value, End_Value and Step_Value are calculated at the beginning of the FOR loop. On the first iteration, Control_Variable is set to Start_Value.

At the beginning of each iteration, the termination condition is tested. If it is satisfied, execution of the loop is complete and the statements after the loop will proceed. If the termination condition is not satisfied, the statements within the FOR…END_FOR construct are executed. At the end of each iteration, the value of Control_Variable is incremented by Step_Value (or 1 if Step_Value is not specified).

The termination condition of a FOR loop depends on the sign of the step value.

Step Value	***Termination Condition***
> 0	Control_Variable > End_Value
< 0	Control Variable < End Value
0	None. A termination condition is never reached and the loop will repeat infinitely.

As with the other iterative statements (WHILE and REPEAT), loop execution can be prematurely halted by an EXIT statement.

To avoid infinitely repeating or unpredictable loops, the following precautions are recommended:

- Do not allow the statement list logic within the FOR loop to modify the control variable.
- Do not use the control variable in logic outside the FOR loop.

Examples

The following code fragment initializes an array of 100 elements starting at %R1000 (given that R1000 is at %R1000) by assigning a value of 10 to all array elements.

```
FOR R1000 := 1 TO 100 DO
    @R1000 := 10;
END_FOR;
```

The following code fragment assigns the values of an I/O point to array elements over ten I/O scans. The last entry is put in the array element with the smallest index.

```
FOR R1000 := 10 TO 1 BY -1 DO
    @R1000 := Input01;
END_FOR;
```

WHILE Statement

The WHILE loop repeatedly executes (iterates) a statement list contained within the WHILE…END_WHILE construct as long as a specified condition is TRUE (1). It checks the condition first, then conditionally executes the statement list. This looping construct is useful when the statement list does not necessarily need to be executed.

Format

```
WHILE  <BooleanExpression>  DO
    <StatementList>;
END_WHILE;
```

Where:

BooleanExpression	Any expression that resolves to a Boolean value.
StatementList	Any set of Structured Text statements.

Operation

If BooleanExpression is FALSE (0), the loop is immediately exited; otherwise, if the BooleanExpression is TRUE (1), the StatementList is executed and the loop repeated. The statement list may never execute, since the Boolean expression is evaluated at the beginning of the loop.

Note: It is possible to create an infinite loop that will cause the watchdog timer to expire. Avoid infinite loops.

Example

The following code fragment increments J by a value of 2 as long as J is less than or equal to 100.

```
WHILE J <= 100 DO
   J := J + 2;
END_WHILE;
```

REPEAT Statement

The REPEAT loop repeatedly executes (iterates) a statement list contained within the REPEAT…END_REPEAT construct until an exit condition is satisfied. It executes the statement list first, then checks for the exit condition. This looping construct is useful when the statement list needs to be executed at least once.

Format

```
REPEAT
    StatementList;
UNTIL  BooleanExpression  END_REPEAT;
```

Where:

BooleanExpression	Any expression that resolves to a Boolean value.
StatementList	Any set of Structured Text statements.

Operation

The StatementList is executed. If the BooleanExpression is FALSE (0), then the loop is repeated; otherwise, if the BooleanExpression is TRUE (1), the loop is exited. The statement list executes at least once, since the BooleanExpression is evaluated at the end of the loop.

Note: It is possible to create an infinite loop that will cause the watchdog timer to expire. Avoid infinite loops.

Example

The following code fragment reads values from an array until a value greater than 5 is found (or the upper bound of the array is reached). Since at least one array value must be read, the REPEAT loop is used. All variables in this example are of type DINT, UINT, or INT.

```
Index :=1;

REPEAT
    Value:= @Index;
    Index:=Index+1;
UNTIL  Value > 5  OR Index >= UpperBound END_REPEAT;
```

ARG_PRES Statement

The ARG_PRES function determines whether an input parameter value was present when the function block instance of the parameter was invoked. This may be necessary if the parameter is optional (pass by value).

This function must be called from a function block instance or a parameterized block.

Format

ARG_PRES (IN :=In1, Q:>Out1, ENO:>Out2);

Where:

In1	Must be an input parameter of the function block that contains the ARG_PRES instruction. Cannot be an array element or structure element. An alias to a parameter should resolve only to the parameter name. Can be a BOOL, DINT, DWORD, INT, REAL, UINT, WORD variable, variable array head name or variable array head name element [000]. Input or output parameter value of a function block instance or a parameterized block
Out2	A BOOL variable. True if the parameter is present, otherwise false.

Note: ENO is an optional BOOL output parameter. If ENO is used in a statement that uses the formal convention, the state of *Out2* is set to 1 (call was successful) or 0 (call failed).

Example

The parameter TempVal is an input to the function block CheckTemp. In the following code fragment, ARG_PRES is used to determine whether a value existed for the parameter TempVal when an instance of CheckTemp was invoked. If TempVal had a value, the BOOL output Temp_Pres is set to 1.

ARG_PRES (TempVal, Temp_Pres);

Exit Statement

The EXIT statement is used to terminate and exit from a loop (FOR, WHILE, REPEAT) before it would otherwise terminate. Program execution resumes with the statement following the loop terminator (END_FOR, END_WHILE, END_REPEAT). An EXIT statement is typically used within an IF statement.

Format

```
EXIT;
```

Where:

ConditionForExiting An expression that determines whether to terminate early.

Example

The following code fragment shows the operation of the EXIT statement. When the variable number equals 10, the WHILE loop is exited and execution continues with the statement immediately following END_WHILE.

```
while (1) do
  a := a + 1;
  IF (a = 10) THEN
    EXIT;
  END_IF;
END_WHILE;
```

Chapter 13 Communications

This chapter describes the Ethernet and Serial communications features of the PACSystems CPU. The following information is included:

Ethernet Communications 13-2

Ethernet Port Pin Assignments 13-2

Serial Communications 13-3

Serial Port Communications Capabilities 13-3

Serial Port Pin Assignments 13-4

Serial Port Baud Rates 13-7

Series 90-70 Communications and Intelligent Option Modules (RX7i only) 13-8

Communications Coprocessor Module (CMM) 13-8

Programmable Coprocessor Module (PCM) 13-9

DLAN/DLAN+ (Drives Local Area Network) Interface 13-10

Ethernet Communications

For details on Ethernet communications for PACSystems, please refer to the following manuals:

TCP/IP Ethernet Communications for PACSystems User's Guide, GFK-2224

PACSystems TCP/IP Communications Station Manager Manual, GFK-2225

Embedded Ethernet Interface

RX7i CPUs have an embedded Ethernet interface that provides TCP/IP communications with other control systems and programming software. These communications use the proprietary SRTP protocol and the standard Modbus/TCP protocol over a four-layer TCP/IP (Internet) stack. The Ethernet interface also supports Ethernet Global Data protocol using UDP (user datagram protocol).

The embedded Ethernet interface has two RJ-45 Ethernet ports. Either or both of these ports may be attached to other Ethernet devices. Each port automatically senses the data rate (10Mbps or 100Mbps), duplex (half duplex or full duplex), and cabling arrangement (straight through or crossover) of the attached link.

Caution

The two ports on the Ethernet Interface must not be connected, directly or indirectly to the same device. The hub or switch connections in an Ethernet network must form a tree, otherwise duplication of packets may result.

10Base-T/100Base-Tx Port Pin Assignments

Pin Number	Signal	Description
1	TD+	Transmit Data +
2	TD-	Transmit Data -
3	RD+	Receive Data +
4	NC	No connection
5	NC	No connection
6	RD-	Receive Data -
7	NC	No connection
8	NC	No connection

Ethernet Interface Modules

The RX7i and RX3i support rack-based Ethernet Interface modules. (These modules are not interchangeable.) For details about the capabilities, installation, and operation of the Ethernet Interface modules, refer to *TCP/IP Ethernet Communications for PACSystems,* GFK-2224 and *Station Manager for PACSystems,* GFK-2225.

Type	Catalog Number	Description
RX7i	IC698ETM001	Ethernet peripheral VME module
RX3i	IC695ETM001	Ethernet peripheral PCI module

Serial Communications

The CPU's independent on-board serial ports are accessed by connectors on the front of the module. Ports 1 and 2 provide serial interfaces to external devices. Port 1 is also used for firmware upgrades. The RX7i CPUs provide a third serial port that is used as the Ethernet station manager port. All serial ports are isolated.

Serial Port Communications Capabilities

Ports 1 and 2 can each be configured for one of the following modes. For details on CPU configuration, refer to chapter 3.

- RTU Slave – The port can be used for the Modbus RTU slave protocol. This mode also permits connection to the port by an SNP master, such as the Winloader utility or the programming software. For details, refer to chapter 14, "Serial I/O, RTU and SNP Protocols."
- Message Mode – The port is available for access by user logic. This enables C language blocks to perform serial port I/O operations via C Runtime Library functions.
- Available – The port is not to be used by the CPU firmware.
- SNP Slave – The port can only be used for the SNP slave protocol. For details, refer to chapter 14, "Serial I/O, RTU and SNP Protocols."
- Serial I/O – The port can be used for general-purpose serial communication through use of COMMREQ functions. For details, refer to chapter 14, "Serial I/O, RTU and SNP Protocols."

Features Supported

Feature	***Port 1 (COM 1)***	***Port 2 (COM 2)***	***Port 3 (Station Mgr) RX7i only***
RTU Slave protocol	Yes	Yes	No
SNP Slave	Yes	Yes	No
Serial I/O – used with COMMREQs	Yes	Yes	No
Firmware Upgrade (Winloader utility)	CPU in STOP/No IO mode	No	No
Message Mode –used only with C blocks (C Runtime Library Functions: serial read, serial write, sscanf, sprintf)	Yes	Yes	No
Station Manager (RX7i only)	No	No	Yes
RS-232	Yes	No	Yes
RS-485	No	Yes	No

Configurable Stop Mode Protocols

You can configure the protocol to be used in Stop mode, based upon the configured Port (Run mode) protocol. The Run/Stop protocol switching is independently configured for each serial port.

The Run mode protocol setting determines which choices are available for Stop mode. If a Stop mode protocol is not selected, the default Stop mode protocol is used. For details, refer to “Port 1 and Port 2 Parameters” in chapter 3.

Serial Port Pin Assignments

Port 1

Port 1 is RS-232 compatible and optocoupler isolated. It has a 9-pin, female, D-sub connector with a standard pin out. This is a DCE (data communications equipment) port that allows a simple straight-through cable to connect with a standard AT-style RS-232 port.

Port 1 RS-232 Signals

Pin Number	Signal Name	Description
1*	NC	No Connection
2	TXD	Transmit Data
3	RXD	Receive Data
4	DSR	Data Set Ready
5	0V	Signal Ground
6	DTR	Data Terminal Ready
7	CTS	Clear To Send
8	RTS	Request to Send
9	NC	No Connection

* Pin 1 is at the bottom right of the connector as viewed from the front of the module.

Port 2

Port 2 is RS-485 compatible and optocoupler isolated. Port 2 has a 15-pin, female D-sub connector. This port does not support the RS-485 to RS-232 adapter (IC690ACC901). This is a DCE port.

Port 2 RS-485 Signals – RX7i CPUs

This port does not supply +5V volts, therefore RS-485 to RS-232 conversion requires a converter that is self-powered.

Pin No.	Signal Name	Description
1*	Shield	Cable Shield Located at the bottom right of the connector as viewed from the front of the module.
2	NC	No Connection
3	NC	No Connection
4	NC	No Connection
5	NC	No Connection
6	RTS(A)	Differential Request to Send
7	0V	Signal Ground
8	CTS(B‘)	Differential Clear To Send
9	RT*	Resistor Termination
10	RD(A‘)*, **	Differential Receive Data
11	RD(B‘)**	Differential Receive Data
12	SD(A)	Differential Send Data
13	SD(B)	Differential Send Data
14	RTS(B’)	Differential Request To Send
15	CTS(A’)	Differential Clear To Send

* To provide termination using the built-in 120 Ω resistor, install a jumper between pins 9 and 10.

** To provide termination using an external resistor, connect a user-supplied resistor across pins 10 and 11.

Port 2 RS-485 Signals – RX3i CPU

Pin No.	Signal Name	Description
1*	Shield	Cable Shield Located at the bottom right of the connector as viewed from the front of the module.
2	NC	No Connection
3	NC	No Connection
4	NC	No Connection
5	+5VDC	Logic Power Provides isolated +5VDC power (300mA maximum) for powering external options.
6	RTS(A)	Differential Request to Send
7	0V	Signal Ground
8	CTS(B‘)	Differential Clear To Send
9	RT*	Resistor Termination
10	RD(A‘)*, **	Differential Receive Data
11	RD(B‘)**	Differential Receive Data
12	SD(A)	Differential Send Data
13	SD(B)	Differential Send Data
14	RTS(B’)	Differential Request To Send
15	CTS(A’)	Differential Clear To Send

* To provide termination using the built-in 120 Ω resistor, install a jumper between pins 9 and 10

** To provide termination using an external resistor, connect a user-supplied resistor across pins 10 and 11.

Port 3 (RX7i only)

Port 3, the Station Manager serial port used by the embedded Ethernet Interface, is RS-232 compatible and isolated. Port 3 has a 9-pin, female, D-connector. This is a DCE port that allows a simple straight-through cable to connect with a standard AT-style RS-232 port. This port contains full use of the standard RS-232 signals for future use with point-to-point protocol (PPP).

Station Manager RS-232 Signals

Pin Number	Signal Name	Description
1*	DCD	Data Carrier Detect
2	TXD	Transmit Data
3	RXD	Receive Data
4	DSR	Data Set Ready
5	0V	Signal Ground
6	DTR	Data Terminal Ready
7	CTS	Clear To Send
8	RTS	Request to Send
9	RI	Ring Indicator

* Pin 1 is at the bottom right of the connector as viewed from the front of the module.

Serial Cable Lengths and Shielding

The connection from a CPU serial port to the serial port on a computer or other serial device requires a serial cable. Maximum cable lengths (the total distance from the CPU to the last device attached to the serial cable) are:

Port	*Cable Length*	*Cable Type*
COM 1/Port 1 (RS-232)	15 meters (50 ft.)	Shielded cable ***required*** for RX3i; Shielded cable optional for RX7i
COM 2/Port 2 (RS-485)	1200 meters (4000 ft.)	Shielded cable ***required*** for all models
STA MGR/Port 3 (RS-232)	15 meters (50 ft.)	Shielded cable optional (RX7i only)

Note: For details on conformance to radiated emissions standards, refer to Appendix A in the following manuals:

PACSystems RX7i Installation Manual, GFK-2223

PACSystems RX3i System Manual, GFK-2314

Serial Port Baud Rates

Protocol	*Port 1 (RS-232)*	*Port 2 (RS-485)*	*Station Mgr (Port 3) (RS-232)*
RTU protocol	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	not supported
Firmware Upgrade via WinLoader	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	Not supported	not supported
Message Mode	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	not supported
SNP Slave	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	not supported
Serial I/O	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K	not supported

Series 90-70 Communications and Intelligent Option Modules

PACSystems RX7i supports the following Series 90-70 communications and intelligent option modules:

- Communications Coprocessor Module (CMM), IC697CMM711
- Programmable Coprocessor Module (PCM), IC697PCM711
- DLAN Interface Module, IC697BEM763

Communications Coprocessor Module (CMM)

PACSystems RX7i CPUs with versions 1.50 and higher support IC697CMM711 modules with firmware versions 4.20 and higher. You must ensure that you are using a valid version of the CMM firmware because the CPU cannot check the CMM's firmware version. (The module's firmware version can be found on a label attached to the EEPROM.)

PACSystems ***does not*** support the following with an IC697CMM711:

- Access to Symbolic variables
- WAIT mode COMMREQs.
- Connecting the programming software to the CPU through the CMM's serial ports.
- Permanent datagrams.

The following restrictions apply when using the IC697CMM711 with PACSystems:

- Access to %W references is partially supported. Only offsets 0—65535 of %W can be accessed via the CMM.
- The Program Name is currently always LDPROG1 for PACSystems.
- Reads and writes beyond currently configured reference table limits will report a minor code error of 90 (REF_OUT_OF_RANGE) instead of F4 (INVALID_PARAMETER) as reported on the Series 90-70.
- In case of ERROR NACK, the Control Program number, privilege level and other piggyback status data will be set to 0.
- PACSystems CPUs return the major/minor type of the 90-70 CPX935 (major type 12, minor type 35) to the CMM scratch pad memory when communicating with a CMM.
- Control Program Number will be returned as 01 in PACSystems instead of FF as reported on the Series 90-70.
- If your RX7i application program needs to access the dual port memory of a CMM, use the BUS READ and WRITE functions. When accessing the CMM, set the Region parameter on the function block to 1. (For the CMM, region 1 is predefined to be the module's entire dual port memory.)

Note: For details on operation of the IC697CMM711, refer to the *Serial Communications User's Manual,* GFK-0582.

Programmable Coprocessor Module (PCM)

PACSystems RX7i CPUs with versions 1.50 and higher support IC697PCM711modules with firmware versions 4.05 and higher. You must ensure that you are using a valid version of the PCM firmware because the CPU cannot check the PCM's firmware version. (The module's firmware version can be found on a label attached to the EEPROM.)

PACSystems ***does not*** support the following with an IC697PCM711:

- Connecting the programming software to the CPU through the PCM's serial ports.
- Access to Symbolic variables.
- WAIT mode COMMREQs.
- The following C functions are not supported:
  - chk_genius_bus
  - chk_genius_device
  - get_cpu_type_rev
  - get_memtype_sizes
  - get_one_rackfault
  - get_rack_slot_faults
- The C function write_dev will not write to "read only" references (%S references, transition bits, and override bits). If this is attempted, the call will fail at run time and return an error code.

The following restrictions apply when using the IC697PCM711 with PACSystems:

- Access to %W references is partially supported. Only offsets 0—65535 of %W can be accessed via the PCM.
- The Program Name is currently always LDPROG1 for PACSystems.
- In case of ERROR NACK, the Control Program number, privilege level and other piggyback status data will be set to 0.
- If an application program running on the PCM accesses the VME bus, the VME addresses being used by that program must be in agreement with the PACSystems RX7i VME address assignments. The PACSystems RX7i VME address assignments are described in the *PACSystems RX7i User's Guide to Integration of VME Modules,* GFK-2235.
- PACSystems CPUs return the major/minor type of the Series 90-70 CPX935 (major type 12, minor type 35) to the PCM scratch pad memory when communicating with a PCM.
- If your RX7i application program needs to access the PCM's dual port memory, use the BUS READ and WRITE functions. When accessing the PCM, set the Region parameter on the function block to 1. (For the PCM, region 1 is predefined to be the module's entire dual port memory.)

Note: For details on operation of the IC697PCM711, refer to *Programmable Coprocessor Module and Support Software,* GFK-0255.

DLAN/DLAN+ (Drives Local Area Network) Interface

PACSystems RX7i CPUs with versions 1.50 and higher support IC697BEM763 modules with firmware versions 3.00 and higher. You must ensure that you are using a valid version of the PCM firmware because the CPU cannot check the DLAN's firmware version. (The module's firmware version can be found on a label attached to the EEPROM.)

If your RX7i application program needs to access the DLAN's dual port memory, use the BUS READ and WRITE functions. When accessing a DLAN module, set the Region parameter on the function block to 1. (For the DLAN module, region 1 is predefined to be the module's entire dual port memory.)

Note: The DLAN Interface module is a specialty module with limited availability. If you have a DLAN system, refer to the *DLAN/DLAN+ Interface Module User's Manual,* GFK-0729 for details.

Chapter

14

Serial I/O, SNP and RTU Protocols

This chapter describes the Serial I/O feature, which can be used to control the read/write activities of CPU serial ports 1 and 2 directly from the application program.

This chapter also contains instructions for using COMM_REQs to configure the CPU serial ports for SNP, RTU, or Serial I/O protocol.

- Configuring Serial Ports Using the COMM_REQ Function
  - RTU Slave/SNP Slave Operation with a Programmer Attached
  - COMM_REQ Command Block for Configuring SNP Protocol
  - COMM_REQ Data Block for Configuring RTU Protocol
  - COMM_REQ Data Block for Configuring Serial I/O
- Serial I/O COMM_REQ Commands
  - Initialize Port
  - Set Up Input Buffer
  - Flush Input Buffer
  - Read Port Status
  - Write Port Control
  - Cancel Operation
  - Autodial
  - Write Bytes
  - Read Bytes
  - Read String
- RTU Slave Protocol
- SNP Slave Protocol

Details of the RTU and SNP protocol are described in the *Serial Communications User's Manual* (GFK-0582).

Configuring Serial Ports Using the COMM_REQ Function

Serial I/O is implemented through the use of Communication Request (COMM_REQ) functions. The operations of the protocol, such as transmitting a character through the serial port or waiting for an input character, are implemented through the COMM_REQ function block.

The COMM_REQ requires that all its command data be placed in the correct order (in a *command block*) in the CPU memory before it is executed. The COMM_REQ should be executed by a contact of a one-shot coil to prevent sending the data multiple times. For details on the operands and command block format used by the COMM_REQ function, refer to chapter 8, “”Instruction Set Reference.”

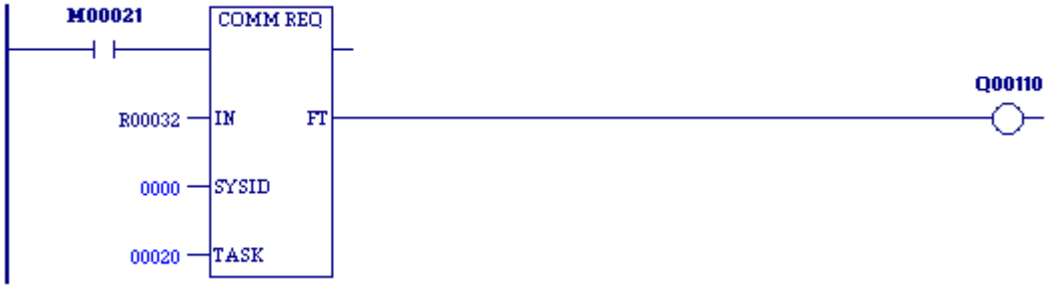
The COMM_REQ uses the following TASKs to specify the port for which the operation is intended:

task 19 for port 1
task 20 for port 2

Note: Because address offsets are stored in a 16-bit word field, the full range of %W memory type cannot be used with COMM_REQs.

COMM_REQ Function Example

In the example, when %M0021 is ON, a Command Block located starting at %R0032 is sent to port 2 (communications task 20) of the CPU (rack 0, slot 0). If an error occurs processing the COMM_REQ, %Q0110 is set.



Timing

If a port configuration COMM_REQ is sent to a serial port that currently has an SNP master (for example, the programmer) connected to it, the COMM_REQ function returns an error code to the COMM_REQ status word.

Sending Another COMM_REQ to the Same Port

After sending a COMM_REQ to configure a serial port, the application program should monitor the COMM_REQ status word to determine when it can begin sending protocol specific COMM_REQs to that port. It is recommended that the application clear the COMM_REQ status word prior to issuing the configuration change. The status word will be set to a nonzero value when the request has been processed.

Invalid Port Configuration Combinations

For the RX3i CPU, the Machine Edition programming software safeguards against the download of hardware configurations that would prevent the programmer from communicating serially with the CPU. This is done because the Ethernet module for the RX3i may not be present or may be removed, in which case a serial connection is required for programmer communications. For the RX7i CPU, the Ethernet port is present on the CPU module, so Ethernet is always available for programmer communications.

COMM_REQ Command Block Parameter Values

The following table lists common parameter values that are used within the COMM_REQ command blocks for configuring a serial port. All values are in decimal.

Parameter	***Values***
Protocol Selector	1 = SNP 3 = RTU 5 = Serial I/O 7 = Message Mode
Data Rate	0 = 300 1 = 600 2 = 1200 3 = 2400 4 = 4800 5 = 9600 6 = 19200 7 = 38400 8 = 57600 9 = 115200
Parity	0 = None 1 = Odd 2 = Even
Flow Control	0 = Hardware [RTS / CTS] 1 = None 2 = Software [XON / XOFF] (Serial I/O only)
Bits Per Character	0 = 7 bits 1 = 8 bits
Stop Bits	0 = 1 stop bit 1 = 2 stop bits
Duplex Mode	0 = 2-wire 1 = 4-wire 2 = 4-wire transmitter always on
Turnaround Delay (SNP only)	0 = none 1 = 10 ms 2 = 100 ms 3 = 500 ms
Timeout (SNP only)	0 = Long (8 sec) 1 = Medium (2 sec) 2 = Short (500 ms) 3 = “None” (200 ms)

Sample COMM_REQ Command Blocks

The following COMM_REQ command blocks provide examples for configuring the various protocols. All values are in decimal unless followed by an H indicating hexadecimal.

Note that an example is not provided for Message Mode, but it can be setup with a command block similar to the one for Serial I/O, with a value of 7 for the protocol selector.

Example COMM_REQ Command Block for Configuring SNP Protocol

	Values	***Meaning***
Address	16	Data Block Length
Address + 1	0 = No Wait (WAIT mode not supported)	WAIT/NOWAIT Flag
Address + 2	0008 = %R, register memory	Status Word Pointer Memory Type
Address + 3	Zero-based number that gives the address of the COMM_REQ status word (for example, a value of 99 gives an address of 100 for the status word)	Status Word Pointer Offset
Address + 4	not used	Idle Timeout Value
Address + 5	not used	Maximum Communication Time
Address + 6	FFF0H	Command Word (serial port setup)
Address + 7	1 = SNP	Protocol
Address + 8	0 = Slave	Port Mode
Address + 9	See “COMM_REQ Command Block Parameter Values” on page 14-3.	Data Rate
Address + 10	0 = None, 1 = Odd, 2 = Even	Parity
Address + 11	not used (SNP always chooses NONE by default)	Flow Control
Address + 12	0 = None, 1 = 10ms, 2 = 100ms, 3 = 500ms	Turnaround Delay
Address + 13	0 = Long, 1 = Medium, 2 = Short, 3 = None	Timeout
Address + 14	not used (SNP always chooses 8 bits by default)	Bits Per Character
Address + 15	0 = 1 Stop Bit, 1 = 2 Stop bits	Stop Bits
Address + 16	not used	Interface
Address + 17	not used (SNP always chooses 4-wire mode by default)	Duplex Mode
Address + 18	user-provided*	Device identifier bytes 1 and 2
Address + 19	user-provided*	Device identifier bytes 3 and 4
Address + 20	user-provided*	Device identifier bytes 5 and 6
Address + 21	user-provided*	Device identifier bytes 7 and 8

* The device identifier for SNP Slave ports is packed into words with the least significant character in the least significant byte of the word. For example, if the first two characters are “A” and “B,” the Address + 18 will contain the hex value 4241.

Example COMM_REQ Data Block for Configuring RTU Protocol

	Values	***Meaning***
Address	13	Data Block Length
Address + 1	0 = No Wait (WAIT mode not supported)	WAIT/NOWAIT Flag
Address + 2	0008 = %R, register memory	Status Word Pointer Memory Type
Address + 3	Zero-based number that gives the address of the COMM_REQ status word (for example, a value of 99 gives an address of 100 for the status word)	Status Word Pointer Offset
Address + 4	not used	Idle Timeout Value
Address + 5	not used	Maximum Communication Time
Address + 6	FFF0H	Command Word (serial port setup)
Address + 7	3 = RTU	Protocol
Address + 8	0 = Slave	Port Mode
Address + 9	See “COMM_REQ Command Block Parameter Values” on page 14-3.	Data Rate
Address + 10	0 = None, 1 = Odd, 2 = Even	Parity
Address + 11	0 = Hardware, 1 = None	Flow Control
Address + 12	not used	Turnaround delay
Address + 13	not used	Timeout
Address + 14	not used (RTU always chooses 8 bits by default)	Bits per Character
Address + 15	not used (RTU always chooses 1 stop bit by default)	Stop Bits
Address + 16	not used	Interface
Address + 17	0 = 2-wire, 1 = 4-wire, 2 = 4-wire transmitter always on	Duplex Mode
Address + 18	Station Address (1-247)	Device Identifier

Example COMM_REQ Data Block for Configuring Serial I/O Protocol

	Values	***Meaning***
Address	12	Data Block Length
Address + 1	0 = No Wait (WAIT mode not supported)	WAIT/NOWAIT Flag
Address + 2	0008 = %R, register memory	Status Word Pointer Memory Type
Address + 3	Zero-based number that gives the address of the COMM_REQ status word (for example, a value of 99 gives an address of 100 for the status word)	Status Word Pointer Offset
Address + 4	not used	Idle Timeout Value
Address + 5	not used	Maximum Communication Time
Address + 6	FFF0H	Command Word (serial port setup)
Address + 7	5 = Serial I/O	Protocol
Address + 8	not used	Port Mode
Address + 9	See “COMM_REQ Command Block Parameter Values” on page 14-3.	Data Rate
Address + 10	0 = None, 1 = Odd, 2 = Even	Parity
Address + 11	0 = Hardware, 1 = None, 2 = Software	Flow Control
Address + 12	not used	Turnaround Delay
Address + 13	not used	Timeout
Address + 14	0=7 bits, 1=8 bits	Bits per Character
Address + 15	0 = 1 stop bit, 1 = 2 stop bits	Stop Bits
Address + 16	not used	Interface
Address + 17	0 = 2-wire, 1 = 4-wire, 2 = 4-wire transmitter always on	Duplex Mode

Calling Serial I/O COMM_REQs from the CPU Sweep

Implementing a serial protocol using Serial I/O COMM_REQs may be restricted by the sweep time. For example, if the protocol requires that a reply to a certain message from the remote device be initiated within 5 ms of receiving the message, this method may not be successful if the sweep time is 5 ms or longer, since timely response is not guaranteed.

Serial I/O protocol is only active when the CPU is in run mode, since it is completely driven by COMM_REQ functions in the application program. When the CPU is stopped, a port configured for Serial I/O will revert to a stop mode protocol. By specifying the stop mode in the port settings of the CPU configuration, the stop mode protocol can be set to either RTU slave or SNP slave; the default is RTU slave.

Compatibility

The COMM_REQ function blocks supported by Serial I/O are not supported by other currently existing protocols (such as SNP slave and RTU slave). Errors are returned if they are attempted for a port configured for one of those protocols.

Status Word for Serial I/O COMM_REQs

A value of 1 is returned in the COMM_REQ status word upon successful completion of the COMM_REQ. Any other value returned is an error code where the low byte is a major error code and the high byte is a minor error code.

Major Error Code	*Description*	
1 (01h)	**Successful Completion** (this is the expected completion value in the COMM_REQ status word).	
12 (0Ch)	**Local error** —Error processing a local command. The minor error code identifies the specific error.	
	2 (02h)	COMM_REQ command is not supported.
	6 (06h)	Invalid PLC memory type specified.
	7 (07h)	Invalid PLC memory offset specified.
	8 (08h)	Unable to access PLC memory.
	12 (0Ch)	COMM_REQ data block length too small.
	14 (0Eh)	COMM_REQ data is invalid.
	15 (0Fh)	Could not allocate system resources to complete COMM_REQ.
13 (0Dh)	**Remote error** — Error processing a remote command. The minor error code identifies the error.	
	2 (02h)	Number of bytes requested to read is greater than input buffer size OR number bytes requested to write is zero or greater than 250 bytes.
	3 (03h)	COMM_REQ data block length is too small. String data is missing or incomplete.
	4 (04h)	Receive timeout awaiting serial reception of data
	6 (06h)	Invalid PLC memory type specified.
	7 (07h)	Invalid PLC memory offset specified.
	8 (08h)	Unable to access PLC memory.
	12 (0Ch)	COMM_REQ data block length too small.
	16 (10h)	Operating system service error. The operating system service used to perform the request has returned an error.
	17 (11h)	Port device error. The port device used to perform the service has detected an error. Either a break was received or a UART Error (parity, framing, overrun) occurred.
	18 (12h)	Request cancelled. The request was terminated before it could complete.
	48 (30h)	Serial output timeout. The serial port was unable to transmit the string. (Could be due to missing CTS signal when the serial port is configured to use hardware flow control.)
14 (0Eh)	**Autodial Error** — An error occurred while attempting to send a command string to an attached external modem. The minor error code identifies the specific error.	
	2 (02h)	The modem command string length exceeds end of reference memory type.
	3 (03h)	COMM_REQ Data Block Length too small. Output command string data missing or incomplete.
	4 (04h)	Serial output timeout. The serial port was unable to transmit the modem autodial output.
	5 (05h)	Response was not received from modem. Check modem and cable.
	6 (06h)	Modem responded with BUSY. Modem is unable to complete the requested connection. The remote modem is already in use; retry the connection request later.
	7 (07h)	Modem responded with NO CARRIER. Modem is unable to complete the requested connection. Check the local and remote modems and the telephone line.
	8 (08h)	Modem responded with NO DIALTONE. Modem is unable to complete the requested connection. Check the modem connections and the telephone line.
	9 (09h)	Modem responded with ERROR. Modem is unable to complete the requested command. Check the modem command string and modem.
	10 (0Ah)	Modem responded with RING, indicating that the modem is being called by another modem. Modem is unable to complete the requested command. Retry the modem command later.
	11 (0Bh)	Unknown response received from the modem. Modem unable to complete the request. Check the modem command string and modem. Response should be CONNECT or OK.

Serial I/O COMM_REQ Commands

The following COMM_REQs are used to implement Serial I/O:

- Local COMM_REQs - do not receive or transmit data through the serial port.
  - Initialize Port (4300)
  - Set Up Input Buffer (4301)
  - Flush Input buffer (4302)
  - Read port status (4303)
  - Write port control (4304)
  - Cancel Operation (4399)
- Remote COMM_REQs - receive and/or transmit data through the serial port.
  - Autodial (4400)
  - Write bytes (4401)
  - Read bytes (4402)
  - Read String (4403)

Overlapping COMM_REQs

Some Serial I/O COMM_REQs must complete execution before another COMM_REQ can be processed. Others can be left pending while others are executed.

COMM_REQS that Must Complete Execution

- Autodial (4400)
- Initialize Port (4300)
- Set Up Input Buffer (4301)
- Flush Input buffer (4302)
- Read port status (4303)
- Write port control (4304)
- Cancel Operation (4399)
- Serial Port Setup (FFF0)

COMM_REQs that can be Pending While Others Execute

The table below shows whether Write Bytes, Read Bytes and Read String COMM_REQs can be pending when other COMM_REQs are executed.

	NEW COMM_REQ										
Currently-pending COMM_REQs	***Autodial (4400)***	***Write Bytes (4401)***	***Initialize Port (4300)***	***Set Up Input Buffer (4301)***	***Flush Input Buffer (4302)***	***Read Port Status (4303)***	***Write Port Control (4304***	***Read Bytes (4402)***	***Read String (4403)***	***Cancel Operation (4399)***	***Serial Port Setup (FFF0)***
Write Bytes (4401)	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Read Bytes (4402)	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No
Read String (4403)	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No

Initialize Port Function (4300)

This function causes a reset command to be sent to the specified port. It also cancels any COMM_REQ currently in progress and flushes the internal input buffer. RTS is set to inactive.

Example Command Block for the Initialize Port Function

	Value (decimal)	***Value (hexadecimal)***	***Meaning***
address	0001	0001	Data block length
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4300	10CC	Initialize port command

Operating Notes

Remote COMM_REQs that are cancelled due to this command executing will return a COMM_REQ status word indicating request cancellation (minor code 12H).

CAUTION

If this COMM_REQ is sent when a Write Bytes (4401) COMM_REQ is transmitting a string from a serial port, transmission is halted. The position within the string where the transmission is halted is indeterminate. In addition, the final character received by the device to which the CPU is sending is also indeterminate.

Set Up Input Buffer Function (4301)

This function is provided for compatibility with legacy Serial I/O applications. In PACSystems releases 5.70 and later, the internal input buffer is always set to 2097 bytes. In earlier PACSystems implementations, the internal input buffer is set to 2K bytes.

The Set Up Input Buffer function returns a success status to the COMM_REQ status word, regardless of the buffer length specified in the command block.

As data is received from the serial port it is placed in the input buffer. If the buffer becomes full, any additional data received from the serial port is discarded and the Overflow Error bit in the Port Status word (See Read Port Status Function) is set.

Retrieving Data from the Buffer

Data can be retrieved from the buffer using the Read String or Read Bytes function. It is not directly accessible from the application program.

If data is not retrieved from the buffer in a timely fashion, some characters may be lost.

Example Command Block for the Set Up Input Buffer Function

	VALUE (decimal)	***VALUE (hexadecimal)***	***MEANING***
address	0002	0002	Data block length
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4301	10CD	Setup input buffer command
address +7	0064	0040	Buffer length (in words)

Flush Input Buffer Function (4302)

This operation empties the input buffer of any characters received through the serial port but not yet retrieved using a read command. All such characters are lost.

Example Command Block for the Flush Input Buffer Function

	VALUE (decimal)	***VALUE (hexadecimal)***	***MEANING***
address	0001	0001	Data block length
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4302	10CE	Flush input buffer command

Read Port Status Function (4303)

This function returns the current status of the port. The following events can be detected:

1. A read request was initiated previously and the required number of characters has now been received or the specified time-out has elapsed.
2. A write request was initiated previously and transmission of the specified number of characters is complete or a time-out has elapsed.

The status returned by the function indicates the event or events that have completed. More than one condition can occur simultaneously, if both a read and a write were initiated previously.

Example Command Block for the Read Port Status Function

	VALUE (decimal)	***VALUE (hexadecimal)***	***MEANING***
address	0003	0003	Data block length
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4303	10CF	Read port status command
address +7	0076	004C	Port status memory type (%M)
address +8	0101	0065	Port status memory offset (%M101)

Port Status

The port status consists of a status word and the number of characters in the input buffer that have not been retrieved by the application (characters which have been received and are available).

word 1	Port status word (see below)
word 2	Characters available in the input buffer

The Port Status Word can be:

Bit	***Name***	***Definition***	***Meaning***	
15	RI	Read In progress	Set	Read Bytes or Read String invoked
			Cleared	Previous Read bytes or String has timed out, been canceled, or finished
14	RS	Read Success	Set	Read Bytes or Read String has successfully completed
			Cleared	New Read Bytes or Read String invoked
13	RT	Read Time-out	Set	Receive timeout occurred during Read Bytes or Read String
			Cleared	New Read Bytes or Read String invoked
12	WI	Write In progress	Set	New Write Bytes invoked
			Cleared	Previously-invoked Write Bytes has timed out, been canceled, or finished
11	WS	Write Success	Set	Previously-invoked Write Bytes has successfully completed
			Cleared	New Write Bytes invoked
10	WT	Write Time-out	Set	Transmit timeout occurred during Write Bytes
			Cleared	New Write Bytes invoked
9	CA	Character Available	Set	Unread characters are in the buffer
			Cleared	No unread characters in the buffer
8	OF	Overflow error	Set	Overflow error occurred on the serial port or internal buffer
			Cleared	Read Port Status invoked
7	FE	Framing Error	Set	Framing error occurred on the serial port
			Cleared	Read Port Status invoked
6	PE	Parity Error	Set	Parity error occurred on the serial port
			Cleared	Read Port Status invoked
5	CT	CTS is active	Set	CTS line on the serial port is active or the serial port does not have a CTS line
			Cleared	CTS line on the serial port is not active
4 - 0	U	Not used; should be 0		

Write Port Control Function (4304)

This function forces RTS for the specified port:

Example Command Block for the Write Port Control Function

	VALUE (decimal)	VALUE (hexadecimal)	MEANING
address	0002	0002	Data block length
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4304	10D0	Write port control command
address +7	xxxx	xxxx	Port control word

Port Control Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTS	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

The Port Control Word can be:

15	RTS	Commanded state of the RTS output 1 = Activates RTS 0 = Deactivates RTS
0-14	U	Unused (should be zero)

Operating Note

For CPU port 2 (RS-485), the RTS signal is also controlled by the transmit driver. Therefore, control of RTS is dependent on the current state of the transmit driver. If the transmit driver is not enabled, asserting RTS with the Write Port Control COMM_REQ will not cause RTS to be asserted on the serial line. The state of the transmit driver is controlled by the protocol and is dependent on the current Duplex Mode of the port. For 2-wire and 4-wire Duplex Mode, the transmit driver is only enabled during transmitting. Therefore, RTS on the serial line will only be seen active on port 2 (configured for 2-wire or 4-wire Duplex Mode) when data is being transmitted. For point-to-point Duplex Mode, the transmit driver is always enabled. Therefore, in point-to-point Duplex Mode, RTS on the serial line will always reflect what is chosen with the Write Port Control COMM_REQ.

Cancel COMM_REQ Function (4399)

This function cancels the current operations in progress. It can be used to cancel both read operations and write operations.

If a read operation is in progress and there are unprocessed characters in the input buffer, those characters are left in the input buffer and available for future reads. The serial port is not reset.

Example Command Block for the Cancel Operation Function

	Value (decimal)	***Value (hexadecimal)***	***Meaning***
address	0002	0002	Data block length (2)
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4399	112F	Cancel operation command
address +7	0001	0001	Transaction type to cancel 1 All operations 2 Read operations 3 Write operations

Operating Notes

Remote COMM_REQs that are cancelled due to this command executing will return a COMM_REQ status word indicating request cancellation (minor code 12H).

Caution

If this COMM_REQ is sent in either Cancel All or Cancel Write mode when a Write Bytes (4401) COMM_REQ is transmitting a string from a serial port, transmission is halted. The position within the string where the transmission is halted is indeterminate. In addition, the final character received by the device to which the CPU is sending is also indeterminate.

Autodial Function (4400)

This feature allows the CPU to automatically dial a modem and send a specified byte string.

To implement this feature, the port must be configured for Serial I/O. After the autodial function is executed and the modem has established a connection, other serial I/O functions (Write Bytes, Set Up Input Buffer, Flush Input Buffer, Read Port Status, Write Port Control, Read Bytes, Read String, and Cancel Operation) can be used.

Example

Pager enunciation can be implemented by three commands, requiring three COMM_REQ command blocks:

Autodial: 04400 (1130h)	Dials the modem.
Write Bytes: 04401 (1131h)	Specifies an ASCII string, from 1 to 250 bytes in length, to send from the serial port.
Autodial: 04400 (1130h)	It is the responsibility of the PLC application program to hang up the phone connection. This is accomplished by reissuing the autodial command and sending the hang up command string.

Autodial Command Block

The Autodial command automatically transmits an Escape sequence that follows the Hayes convention. If you are using a modem that does not support the Hayes convention, you may be able to use the Write Bytes command to dial the modem.

Examples of commonly used command strings for Hayes-compatible modems are listed below:

Command String	***Length***	***Function***
ATDP15035559999<CR>	16 (10h)	Pulse dial the number 1-503-555-9999
ATDT15035559999<CR>	16 (10h)	Tone dial the number 1-503-555-9999
ATDT9,15035559999<CR>	18 (12h)	Tone dial using outside line with pause
ATH0<CR>	5 (05h)	Hang up the phone
ATZ <CR>	4 (04h)	Restore modem configuration to internally saved values

Sample Autodial Command Block

This COMM_REQ command block dials the number 234-5678 using a Hayes-compatible modem.

Word	*Definition*	*Values*
1	0009h	CUSTOM data block length (includes command string)
2	0000h	NOWAIT mode
3	0008h	Status word memory type (%R)
4	0000h	Status word address minus 1 (Register 1)
5	0000h	not used
6	0000h	not used
7	04400 (1130h)	Autodial command number
8	00030 (001Eh)	Modem response timeout (30 seconds)
9	0012 (000Ch)	Number of bytes in command string
10	5441h	A (41h), T (54h)
11	5444h	D (44h), T (54h)
12	3332h	Phone number: 2 (32h), 3 (33h)
13	3534h	4 (34h), 5 (35h)
14	3736h	6 (36h), 7 (37h)
15	0D38h	8 (38h) <CR> (0Dh)

Write Bytes Function (4401)

This operation can be used to transmit one or more characters to the remote device through the specified serial port. The character(s) to be transmitted must be in a word reference memory . They should not be changed until the operation is complete.

Up to 250 characters can be transmitted with a single invocation of this operation. The status of the operation is not complete until all of the characters have been transmitted or until a timeout occurs (for example, if hardware flow control is being used and the remote device never enables the transmission).

Example Command Block for the Write Bytes Function

	Value (decimal)	***Value (hexadecimal)***	***Meaning***
address	0006	0006	Data block length (includes characters to send)
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4401	1131	Write bytes command
address +7	0030	001E	Transmit time-out (30 seconds). See note below.
address +8	0005	0005	Number of bytes to write
address +9	25960	6568	‘h’ (68h), ‘e’ (65h)
address +10	27756	6C6C	‘l’ (6Ch), ‘l’ (6Ch)
address +11	0111	006F	‘o’ (6Fh)

Although printable ASCII characters are used in this example, there is no restriction on the values of the characters that can be transmitted.

Operating Notes

Specifying zero as the Transmit time-out sets the time-out value to the amount of time actually needed to transmit the data, plus 4 seconds.

Caution

If an Initialize Port (4300) COMMEQ is sent or a Cancel Operation (4399) COMM_REQ is sent in either Cancel All or Cancel Write mode while this COMM_REQ is transmitting a string from a serial port, transmission is halted. The position within the string where the transmission is halted is indeterminate. In addition, the final character received by the device the CPU is sending to is also indeterminate.

Read Bytes Function (4402)

This function causes one or more characters to be read from the specified port. The characters are read from the internal input buffer and placed in the specified input data area. The function returns both the number of characters retrieved and the number of unprocessed characters still in the input buffer. If zero characters of input are requested, only the number of unprocessed characters in the input buffer is returned.

If insufficient characters are available to satisfy the request and a non-zero value is specified for the number of characters to read, the status of the operation is not complete until either sufficient characters have been received or the time-out interval expires. In either of those conditions, the port status indicates the reason for completion of the read operation. The status word is not updated until the read operation is complete (either due to timeout or when all the data has been received).

If the time-out interval is set to zero, the COMM_REQ remains pending until it has received the requested amount of data, or until it is cancelled.

If this COMM_REQ fails for any reason, no data is returned to the input data area. Any data that has not been read from the internal input buffer remains and it can be retrieved with a subsequent read request.

Example Command Block for the Read Bytes Function

	Value (decimal)	***Value (hexadecimal)***	***Meaning***
address	0005	0005	Data block length
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4402	1132	Read bytes command
address +7	0030	001E	Read time-out (30 seconds)
address +8	0005	0005	Number of bytes to read
address +9	0008	0008	Input data memory type (%R).
address +10	0100	0064	Input data memory address (%R0100)

Return Data Format for the Read Bytes Function

The return data consists of the number of characters actually read, the number of characters still available in the input buffer after the read is complete (if any), and the actual input characters.

Address	Number of characters actually read
Address + 1	Number of characters still available in the input buffer, if any
Address + 2	first two characters (first character is in the low byte)
Address + 3	third and fourth characters (third character is in the low byte)
Address + n	subsequent characters

Operating Notes for Read Bytes

If the input data memory type parameter is specified to be a word memory type, and if an odd number of bytes are actually received, then the high byte of the last word to be written with the received data is left unchanged.

As data is received from the serial port it is placed in the internal input buffer. If the buffer becomes full, then any additional data received from the serial port is discarded and the Overflow Error bit in the Port Status word (See Read Port Status Function) is set.

Read String Function (4403)

This function causes characters to be read from the specified port until a specified terminating character is received. The characters are read from the internal input buffer and placed in the specified input data area.

The function returns both the number of characters retrieved and the number of unprocessed characters still in the input buffer. If zero characters of input are requested, only the number of unprocessed characters in the input buffer are returned.

If the terminating character is not in the input buffer, the status of the operation is not complete until either the terminating character has been received or the time-out interval expires. In either of those conditions, the port status indicates the reason for completion of the read operation.

If the time-out interval is set to zero, the COMM_REQ remains pending until it has received the requested string, terminated by the specified end character.

If this COMM_REQ fails for any reason, no data is returned to the input data area. Any data that has not been read from the internal input buffer remains, and it can be retrieved with a subsequent read request.

Example Command Block for the Read String Function

	Value (decimal)	***Value (hexadecimal)***	***Meaning***
address	0005	0005	Data block length
address +1	0000	0000	NOWAIT mode
address +2	0008	0008	Status word memory type (%R)
address +3	0000	0000	Status word address minus 1 (%R0001)
address +4	0000	0000	Not used
address +5	0000	0000	Not used
address +6	4403	1133	Read string command
address +7	0030	001E	Read time-out (30 seconds)
address +8	0013	000D	Terminating character (carriage return): must be between 0 and 255 (0xFF), inclusive
address +9	0008	0008	Input data memory type (%R)
address +10	0100	0064	Input data memory address (%R0100)

Return Data Format for the Read String Function

The return data consists of the number of characters actually read, the number of characters still available in the input buffer after the read is complete (if any), and the actual input characters:

Address	Number of characters actually read
Address + 1	Number of characters still available in the input buffer, if any
Address + 2	first two characters (first character is in the low byte)
Address + 3	third and fourth characters (third character is in the low byte)
Address + n	subsequent characters

Operating Notes for Read String

If the input data memory type parameter is specified to be a word memory type, and if an odd number of bytes are actually received, then the high byte of the last word to be written with the received data is left unchanged.

As data is received from the serial port it is placed in the internal input buffer. If the buffer becomes full, then any additional data received from the serial port is discarded and the Overflow Error bit in the Port Status word (See Read Port Status Function) is set.

RTU Slave Protocol

RTU protocol is a query-response protocol used for communication between the RTU device and a host computer, which is capable of communicating using RTU protocol. The host computer is the master device and it transmits a query to a RTU slave, which responds to the master. The RTU slave device cannot query; it can only respond to the master. A PACSystems CPU can only function as an RTU slave.

The RTU data transferred consists of 8-bit binary characters with an optional parity bit. No control characters are added to the data block; however, an error check (Cyclic Redundancy Check) is included as the final field of each query and response to ensure accurate transmission of data.

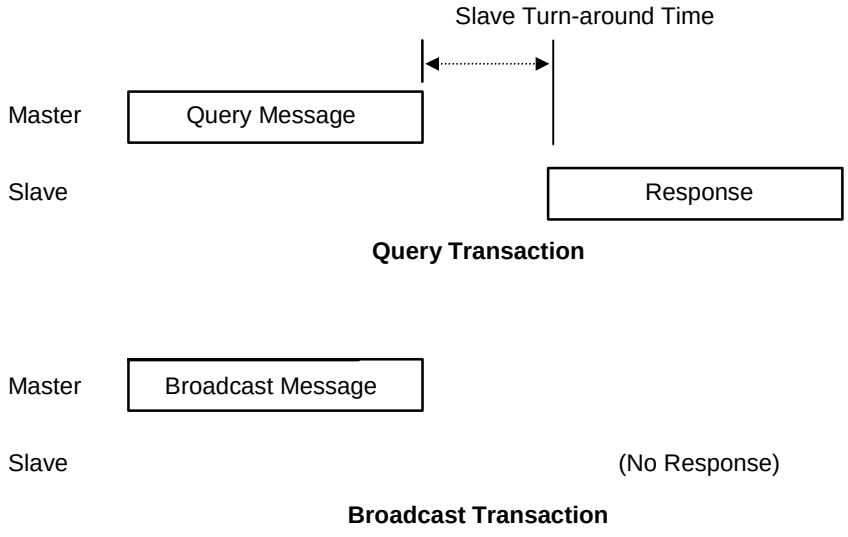
Note: You should avoid using station address 1 for any other Modbus slave in a PACSystems control system because the default station address for the PACSystems CPU is 1. The CPU uses the default address in two situations:

1. If you power up without a configuration, the default station address of 1 is used.
2. When the Port Mode parameter is set to Message Mode, and Modbus becomes the protocol in stop mode, the station address defaults to 1, unless you specify a stop mode for the port in the CPU configuration, and then change the station address to be used for stop mode.

In either of these situations, if you have a slave configured with a station address of 1, confusion may result when the PACSystems CPU responds to requests intended for that slave.

Message Format

The general formats for RTU message transfers are shown below:



RTU Message Transfers

The master device begins a data transfer by sending a query or broadcast request message. A slave completes that data transfer by sending a response message if the master sent a query message addressed to it. No response message is sent when the master sends a broadcast request.

RTU Slave Turnaround Time

The time between the end of a query and the beginning of the response to that query is called the slave turnaround time. The turnaround time of a PACSystems slave depends on the Controller Communications Window time and the sweep time of the PACSystems. RTU requests are processed only in the Controller Communications Window. In Normal sweep mode, the Controller Communications Window occurs once per sweep. Because the sweep time on PACSystems can be up to 2.5 seconds, the time to process an RTU request could be up to 2.5 seconds. Another factor is the Controller Communications Window time allowed in Hardware Configuration. If you configure a very small Controller Communications Window, the RTU request may not be completed in one sweep, causing RTU processing to require multiple sweeps. For details on CPU window modes, refer to chapter 5.

Message Types

The RTU protocol has four message types: query, normal response, error response, and broadcast.

Query

The master sends a message addressed to a single slave.

Normal Response

After the slave performs the function requested by the query, it sends back a normal response for that function. This indicates that the request was successful.

Error Response

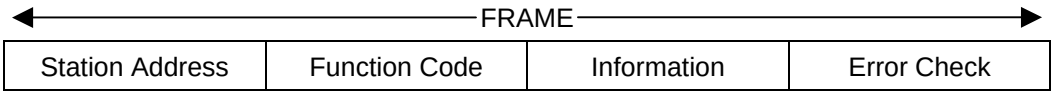
The slave receives the query, but cannot perform the requested function. The slave sends back an error response that indicates the reason the request could not be processed. (No error message will be sent for certain types of errors. For more information see “Communication Errors.”)

Broadcast

The master sends a message addressed to all of the slaves by using address 0. All slaves that receive the broadcast message perform the requested function. This transaction is ended by a time-out within the master.

Message Fields

The message fields for a typical message are shown in the figure below, and are explained in the following sections.



Station Address

The station address is the address of the slave station selected for this data transfer. It is one byte in length and has a value from 0 to 247 inclusive. An address of 0 selects all slave stations, and indicates that this is a broadcast message. An address from 1 to 247 selects a slave station with that station address.

Function Code

The function code identifies the command being issued to the station. It is one byte in length and is defined for the values 0 to 255 as follows:

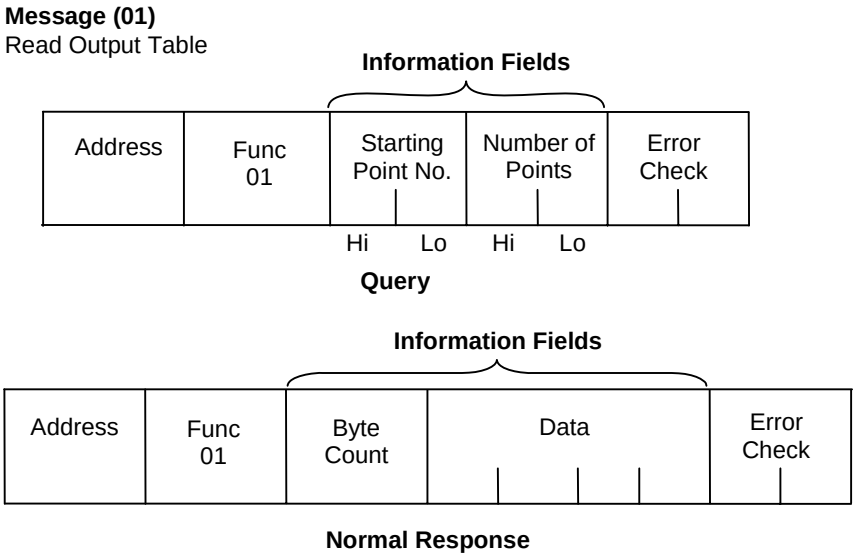
Function Code	***Description***
0	Illegal Function
1	Read Output Table
2	Read Input Table
3	Read Registers
4	Read Analog Input
5	Force Single Output
6	Preset Single Register
7	Read Exception Status
8	Loopback Maintenance
9-14	Unsupported Function
15	Force Multiple Outputs
16	Preset Multiple Registers
17	Report Device Type
18–21	Unsupported Function
22	Mask Write 4x Register
23	Read/Write 4x Registers
24–66	Unsupported Function
67	Read Scratch Pad Memory
68-127	Unsupported Function
128-255	Reserved for Exception Responses

Information Fields

All message fields, other than the Station Address field, Function Code field, and Error Check field are called, generically, “information” fields. Information fields contain additional information required to specify or respond to a requested function. Different types of messages have different types or numbers of information fields. (Details on information fields for each message type and function code are found in “Message Descriptions,” page 14-32) Some messages (Message 07 Query and Message 17 Query) do not have information fields.

Examples

As shown in the following figure, the information fields for message *READ OUTPUT TABLE (01) Query* consist of the Starting Point No. field and Number of Points field. The information fields for message *READ OUTPUT TABLE (01) Response* consist of the Byte Count field and Data field.



Information Field Examples

Some information fields include entries for the range of data to be accessed in the RTU slave.

Note: Data addresses are 0-based. This means you will need to subtract 1 from the actual address when specifying it in the RTU message. For message *(01) READ OUTPUT TABLE Query,* used in the example above, you would specify a starting data address in the Starting Point No. field. To specify %Q0001 as the starting address, you would place the address %Q0000 in this field. Also, the value placed in the Number of Points field determines how many %Q bits are read, starting with address %Q0001. For example:

- Starting Point No. field = %Q0007, so the starting address is %Q0008.
- Number of Points field = 16 (0010h), so addresses %Q0008 through %Q0023 will be read.

Error Check Field

The error check field is two bytes in length and contains a cyclic redundancy check (CRC-16) code. Its value is a function of the contents of the station address, function code, and information field. The details of generating the CRC-16 code are described in “Cyclic Redundancy Check (CRC) on page 14-28” Note that the information field is variable in length. To properly generate the CRC-16 code, the length of frame must be determined. To calculate the length of a frame for each of the defined function codes, see “Calculating the Length of Frame” on page 14-31.

Message Length

Message length varies with the type of message and amount of data to be sent. Information for determining message length for individual messages is found in "Message Descriptions."

Character Format

A message is sent as a series of characters. Each byte in a message is transmitted as a character. The illustration below shows the character format. A character consists of a start bit (0), eight data bits, an optional parity bit, and one stop bit (1). Between characters the line is held in the 1 state.

		MSB			Data Bits				LSB	
10	9	8	7	6	5	4	3	2	1	0
Stop	Parity (optional)									Start

Message Termination

Each station monitors the time between characters. When a period of three character times elapses without the reception of a character, the end of a message is assumed. The reception of the next character is assumed to be the beginning of a new message. The end of a frame occurs when the first of the following two events occurs:

- The number of characters received for the frame is equal to the calculated length of the frame.
- A length of 4 character times elapses without the reception of a character.

Timeout Usage

Timeouts are used on the serial link for error detection, error recovery, and to prevent the missing of the end of messages and message sequences. Note that although the module allows up to three character transmission times between each character in a message that it receives, there is no more than half a character time between each character in a message that the module transmits. After sending a query message, the master should wait an appropriate amount of time for slave turnaround before assuming that the slave did not respond to the request. Slave turnaround time is affected by the Controller Communications Window time and the CPU sweep time, as described in "RTU Slave Turnaround Time" on page 14-24.

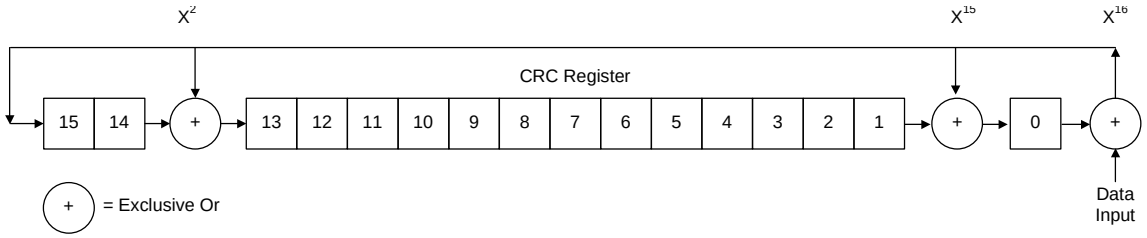
Cyclic Redundancy Check (CRC)

The CRC is one of the most effective systems for checking errors. The CRC consists of two check characters generated at the transmitter and added at the end of the transmitted data characters. Using the same method, the receiver generates its own CRC for the incoming data and compares it to the CRC sent by the transmitter to ensure proper transmission. A complete mathematic derivation for the CRC is not given in this section. This information can be found in a number of texts on data communications. The essential steps that should be understood in calculating the CRC are as follows:

- The number of bits in the CRC multiplies the data bits that make up the message.
- The resulting product is then divided by the generating polynomial (using modulo 2 with no carries). The CRC is the remainder of this division.
- Disregard the quotient and add the remainder (CRC) to the data bits and transmit the message with CRC.
- The receiver then divides the message plus CRC by the generating polynomial and if the remainder is 0, the transmission was transmitted without error.

A generating polynomial is expressed algebraically as a string of terms in powers of X such as $X^3 + X^2 + X^0$ (or 1) which can in turn be expressed as the binary number 1101. A generating polynomial could be any length and contain any pattern of 1s and 0s as long as both the transmitter and receiver use the same value. For optimum error detection, however, certain standard generating polynomials have been developed. RTU protocol uses the polynomial $X^{16} + X^{15} + X^2 + 1$ which in binary is 1 1000 0000 0000 0101. The CRC this polynomial generates is known as CRC-16.

The discussion above can be implemented in hardware or software. One hardware implementation involves constructing a multi-section shift register based on the generating polynomial.



Cyclic Redundancy Check Register

To generate the CRC, the message data bits are fed to the shift register one at a time. The CRC register contains a preset value. As each data bit is presented to the shift register, the bits are shifted to the right. The LSB is XORed with the data bit and the result is: XORed with the old contents of bit 1 (the result placed in bit 0), XORed with the old contents of bit 14 (and the result placed in bit 13), and finally, it is shifted into bit 15. This process is repeated until all data bits in a message have been processed. Software implementation of the CRC-16 is explained in the next section.

Calculating the CRC-16

The pseudo code for calculation of the CRC-16 is given below.

Preset byte count for data to be sent.

Initialize the 16-bit remainder (CRC) register to all ones.

XOR the first 8-bit data byte with the high order byte of the 16-bit CRC register. The result is the current CRC.

INIT SHIFT: Initialize the shift counter to 0.

SHIFT Shift the current CRC register 1 bit to the right.

Increment shift count.

Is the bit shifted out to the right (flag) a 1 or a 0?

If it is a 1, XOR the generating polynomial with the current CRC.

If it is a 0, continue.

Is shift counter equal to 8?

If NO, return to SHIFT.

If YES, increment byte count.

Is byte count greater than the data length?

If NO, XOR the next 8-bit data byte with the current CRC and go to INIT SHIFT.

If YES, add current CRC to end of data message for transmission and exit.

When the message is transmitted, the receiver performs the same CRC operation on all the data bits and the transmitted CRC. If the information is received correctly the resulting remainder (receiver CRC) is 0.

Sample CRC-16 Calculation

The RTU device transmits the rightmost byte (of registers or discrete data) first. The first bit of the CRC-16 transmitted is the MSB. Therefore, in the example the MSB of the CRC polynomial is to the extreme right. The X^{16} term is dropped because it affects only the quotient (which is discarded) and not the remainder (the CRC characters). The generating polynomial is therefore 1010 0000 0000 0001. The remainder is initialized to all 1s.

In this example, the CRC-16 is calculated for RTU message, Read Exception Status 07. The message format is as follows:

Address	Function	CRC-16
01	07	

In this example, device number 1 (address 01) is queried. You need to know the amount of data to be transmitted and this information can be found for every message type in “Calculating the Length of Frame.” For this message the data length is 2 bytes.

Transmitter CRC-16 Algorithm

	MSB[2]		LSB[2]		Flag
Initial Remainder	1111	1111	1111	1111	
XOR 1st data byte	0000	0000	0000	0001	
Current CRC	1111	1111	1111	1111	
Shift 1	0111	1111	1111	1111	0
Shift 2	0011	1111	1111	1111	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1001	1111	1111	1110	
Shift 3	0100	1111	1111	1111	0
Shift 4	0010	0111	1111	1111	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1000	0111	1111	1110	
Shift 5	0100	0011	1111	1111	0
Shift 6	0010	0001	1111	1111	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1000	0001	1111	1110	
Shift 7	0100	0000	1111	1111	0
Shift 8	0010	0000	0111	1111	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1000	0000	0111	1110	
XOR 2nd data byte	0000	0000	0000	0111	
Current CRC	1000	0000	0111	1001	
Shift 1	0100	0000	0011	1100	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1110	0000	0011	1101	
Shift 2	0111	0000	0001	1110	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1101	0000	0001	1111	
Shift 3	0110	1000	0000	1111	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1100	1000	0000	1110	
Shift 4	0110	0100	0000	0111	0
Shift 5	0011	0010	0000	0011	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1001	0010	0000	0010	
Shift 6	0100	1001	0000	0001	0
Shift 7	0010	0100	1000	0000	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Current CRC	1000	0100	1000	0001	
Shift 8	0100	0010	0100	0000	1
XOR Gen. Polynomial	1010	0000	0000	0001	
Transmitted CRC	1110	0010	0100	0001	
	E	2	4	1	

Receiver[1] CRC-16 Algorithm

	MSB[2]		LSB[2]		Flag
Rcvr CRC after data	1110	0010	0100	0001	
XOR 1st byte Trns CRC	0000	0000	0100	0001	
Current CRC	1110	0010	0000	0000	
Shift 1	0111	0001	0000	0000	0
Shift 2	0011	1000	1000	0000	0
Shift 3	0001	1100	0100	0000	0
Shift 4	0000	1110	0010	0000	0
Shift 5	0000	0111	0001	0000	0
Shift 6	0000	0011	1000	1000	0
Shift 7	0000	0001	1100	0100	0
Shift 8	0000	0000	1110	0010	0
XOR 2nd byte trns CRC	0000	0000	1110	0010	
Current CRC	0000	0000	0000	0000	
Shift 1-8 yields	0000	0000	0000	0000	

All errors for receiver final CRC-16 indicates transmission correct.

1. The receiver processes incoming data through the same CRC algorithm as the transmitter. The example for the receiver starts at the point after all the data bits but not the transmitted CRC have been received correctly. Therefore, the receiver CRC should be equal to the transmitted CRC at this point. When this occurs, the output of the CRC algorithm will be zero indicating that the transmission is correct.

   The transmitted message with CRC would then be:

Address	***Function***	***CRC–16***	
01	07	41	E2

2. The MSB and LSB references are to the data bytes only, not the CRC bytes. The CRC MSB and LSB order are the reverse of the data byte order.

Calculating the Length of Frame

To generate the CRC-16 for any message, the message length must be known. The length for all types of messages can be determined from the table below.

RTU Message Length

Function Code And Name		***Query or Broadcast Message Length Less CRC Code***	***Response Message Length Less CRC Code***
0		Not Defined	Not Defined
1	Read Output Table	6	3 + 3rd byte*
2	Read Input Table	6	3 + 3rd byte*
3	Read Registers	6	3 + 3rd byte*
4	Read Analog Input	6	3 + 3rd byte*
5	Force Single Output	6	6
6	Preset Single Register	6	6
7	Read Exception Status	2	3
8	Loopback/Maintenance	6	6
9-14		Not Defined	Not Defined
15	Force Multiple Outputs	7 + 7th byte*	6
16	Preset Multiple Registers	7 + 7th byte*	6
17	Report Device Type	2	8
18-21		Not Defined	Not Defined
22	Mask Write 4x Registers	8	8
23	Read/Write 4x Registers	13+byte 11*	5+byte 3*
24–66		Not Defined	Not Defined
67	Read Scratch Pad	6	3 + 3rd byte*
68-127		Not Defined	Not Defined
128-255		Not Defined	3

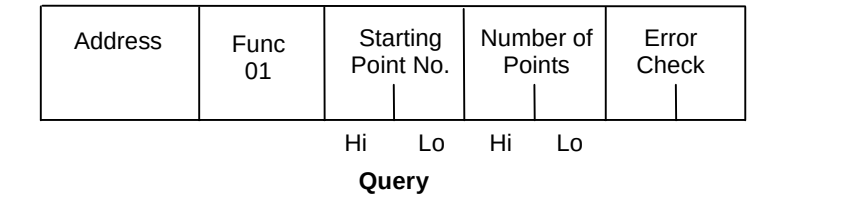
*The value of this byte is the number of bytes contained in the data being transmitted.

RTU Message Descriptions

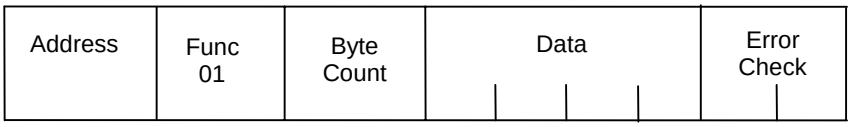
This section presents the format and fields for each RTU message.

Message (01): Read Output Table

Format:



Query



Normal Response

Query:

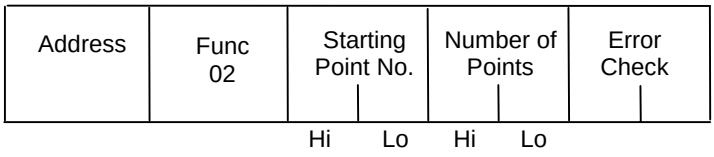
- An address of 0 is not allowed because this cannot be a broadcast request.
- The function code is 01.
- The starting point number is two bytes in length and may be any value less than the highest output point number available in the attached CPU. The starting point number is equal to one less than the number of the first output point returned in the normal response to this request.
- The number of points value is two bytes in length. It specifies the number of output points returned in the normal response. The sum of the starting point value and the number of points value must be less than or equal to the highest output point number available in the attached CPU. The high order byte of the starting point number and number of bytes fields is sent as the first byte. The low order byte is the second byte in each of these fields.

Response:

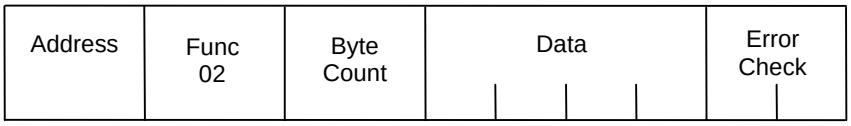
- The byte count is a binary number from 1 to 256 (0 = 256). It is the number of bytes in the normal response following the byte count and preceding the error check.
- The data field of the normal response is packed output status data. Each byte contains 8 output point values. The least significant bit (LSB) of the first byte contains the value of the output point whose number is equal to the starting point number plus one. The values of the output points are ordered by number starting with the LSB of the first byte of the data field and ending with the most significant bit (MSB) of the last byte of the data field. If the number of points is not a multiple of 8, the last data byte contains zeros in one to seven of its highest order bits.

Message (02): Read Input Table

Format:



Query



Normal Response

Query:

- An address of 0 is not allowed as this cannot be a broadcast request.
- The function code is 02.
- The starting point number is two bytes in length and may be any value less than the highest input point number available in the attached CPU. The starting point number is equal to one less than the number of the first input point returned in the normal response to this request.
- The number of points value is two bytes in length. It specifies the number of input points returned in the normal response. The sum of the starting point value and the number of points value must be less than or equal to the highest input point number available in the attached CPU. The high order byte of the starting point number and number of bytes fields is sent as the first byte. The low order byte is the second byte in each of these fields.

Response:

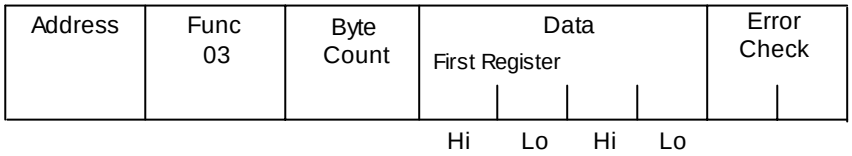
- The byte count is a binary number from 1 to 256 (0 = 256). It is the number of bytes in the normal response following the byte count and preceding the error check.
- The data field of the normal response is packed input status data. Each byte contains 8 input point values. The least significant bit (LSB) of the first byte contains the value of the input point whose number is equal to the starting point number plus one. The values of the input points are ordered by number starting with the LSB of the first byte of the data field and ending with the most significant bit (MSB) of the last byte of the data field. If the number of points is not a multiple of 8, then the last data byte contains zeros in one to seven of its highest order bits.

Message (03): Read Registers

Format:



Query



Normal Response

Query:

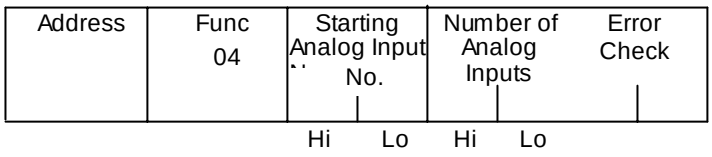
- An address of 0 is not allowed as this request cannot be a broadcast request.
- The function code is equal to 3.
- The starting register number is two bytes in length. The starting register number may be any value less than the highest register number available in the attached CPU. It is equal to one less than the number of the first register returned in the normal response to this request.
- The number of registers value is two bytes in length. It must contain a value from 1 to 125 inclusive. The sum of the starting register value and the number of registers value must be less than or equal to the highest register number available in the attached CPU. The high order byte of the starting register number and number of registers fields is sent as the first byte in each of these fields. The low order byte is the second byte in each of these fields.

Response:

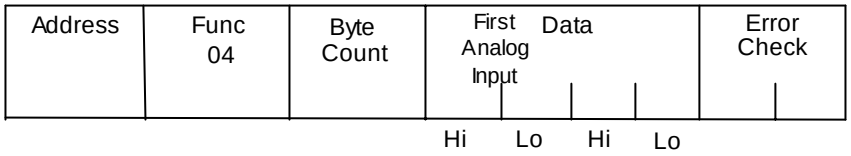
- The byte count is a binary number from 2 to 250 inclusive. It is the number of bytes in the normal response following the byte count and preceding the error check. Note that the byte count is equal to two times the number of registers returned in the response. A maximum of 250 bytes (125) registers is set so that the entire response can fit into one 256 byte data block.
- The registers are returned in the data field in order of number with the lowest number register in the first two bytes and the highest number register in the last two bytes of the data field. The number of the first register in the data field is equal to the starting register number plus one. The high order byte is sent before the low order byte of each register.

Message (04): Read Analog Inputs

Format:



Query



Normal Response

Query:

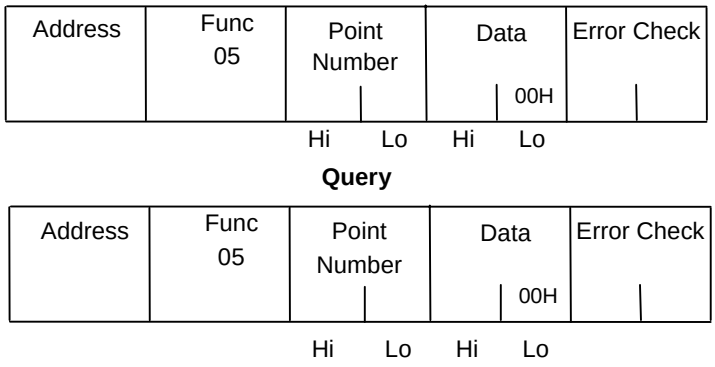
- An address of 0 is not allowed as this request cannot be a broadcast request.
- The function code is equal to 4.
- The starting analog input number is two bytes in length. The starting analog input number may be any value less than the highest analog input number available in the attached CPU. It is equal to one less than the number of the first analog input returned in the normal response to this request.
- The number of analog inputs value is two bytes in length. It must contain a value from 1 to 125 inclusive. The sum of the starting analog input value and the number of analog inputs value must be less than or equal to the highest analog input number available in the at-attached CPU. The high order byte of the starting analog input number and number of analog input fields is sent as the first byte in each of these fields. The low order byte is the second byte in each of these fields.

Response:

- The byte count is a binary number from 2 to 250 inclusive. It is the number of bytes in the normal response following the byte count and preceding the error check. Note that the byte count is equal to two times the number of analog inputs returned in the response. A maximum of 250 bytes (125) analog inputs is set so that the entire response can fit into one 256 byte data block.
- The analog inputs are returned in the data field in order of number with the lowest number analog input in the first two bytes and the highest number analog input in the last two bytes of the data field. The number of the first analog input in the data field is equal to the starting analog input number plus one. The high order byte is sent before the low order byte of each analog input.

Message (05): Force Single Output

Format:



Query

Normal Response

Query:

- An address of 0 indicates a broadcast request. All slave stations process a broadcast re-quest and no response is sent.
- The function code is equal to 05.
- The point number field is two bytes in length. It may be any value less than the highest output point number available in the attached CPU. It is equal to one less than the number of the output point to be forced on or off.
- The first byte of the data field is equal to either 0 or 255 (FFH). The output point specified in the point number field is to be forced off if the first data field byte is equal to 0. It is to be forced on if the first data field byte is equal to 255 (FFH). The second byte of the data field is always equal to zero.

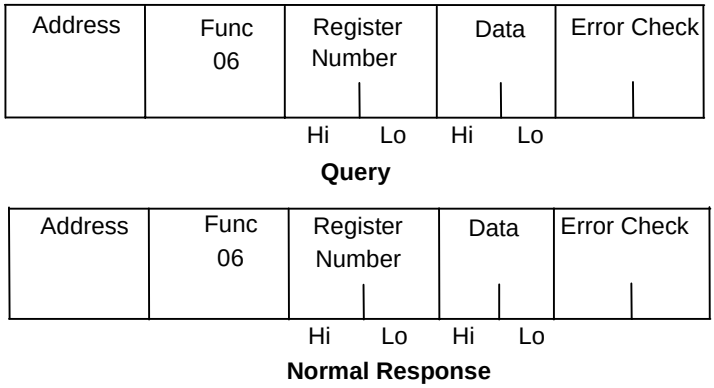
Response:

- The normal response to a force single output query is identical to the query.

Note: The force single output request is not an output override command. The output specified in this request is ensured to be forced to the value specified only at the beginning of one sweep of the user logic.

Message (06): Preset Single Register

Format:



Query:

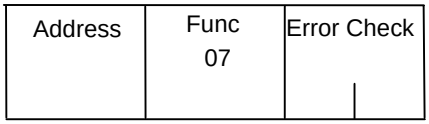
- An address 0 indicates a broadcast request. All slave stations process a broadcast request and no response is sent.
- The function code is equal to 06.
- The register number field is two bytes in length. It may be any value less than the highest register available in the attached CPU. It is equal to one less than the number of the register to be preset.
- The data field is two bytes in length and contains the value that the register specified by the register number field is to be preset to. The first byte in the data field contains the high order byte of the preset value. The second byte in the data field contains the low order byte.

Response:

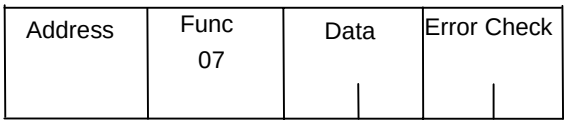
- The normal response to a preset single register query is identical to the query.

Message (07): Read Exception Status

Format:



Query



Normal Response

Query:

This query is a short form of request for the purpose of reading the first eight output points.

- An address of zero is not allowed as this cannot be a broadcast request.
- The function code is equal to 07.

Response:

- The data field of the normal response is one byte in length and contains the states of output points one through eight. The output states are packed in order of number with output point one's state in the least significant bit and output point eight's state in the most significant bit.

Message (08): Loopback/Maintenance (General)

Format:

Address	Func 08	Diagnostic Code 0, 1, or 4	Data DATA 1	DATA 1	Error Check

Query

Address	Func 08	Diagnostic Code 0, 1, or 4	Data DATA 1	DATA 1	Error Check

Normal Response

Query:

- The function code is equal to 8.
- The diagnostic code is two bytes in length. The high order byte of the diagnostic code is the first byte sent in the diagnostic code field. The low order byte is the second byte sent. The loopback/maintenance command is defined only for diagnostic codes equal to 0, 1, or 4. All other diagnostic codes are reserved.
- The data field is two bytes in length. The contents of the two data bytes are defined by the value of the diagnostic code.

Response:

- See descriptions for individual diagnostic codes.

Diagnostic Return Query Data Request (Loopback/Maintenance Code 00):

- An address of 0 is not allowed for the return query data request.
- The values of the two data field bytes in the query are arbitrary.
- The normal response is identical to the query.
- The values of the data bytes in the response are equal to the values sent in the query.

Diagnostic Initiate Communication Restart Request (Loopback/Maintenance Code 01):

- An address of 0 indicates a broadcast request. All slave stations process a broadcast re-quest and no response is sent.
- This request disables the listen-only mode (enables responses to be sent when queries are received so that communications can be restarted).
- The value of the first byte of the data field (DATA1) must be 0 or FF. Any other value will cause an error response to be sent. The value of the second byte of the data field (DATA2) is always equal to 0.
- The normal response to an Initiate Communication Restart query is identical to the query.

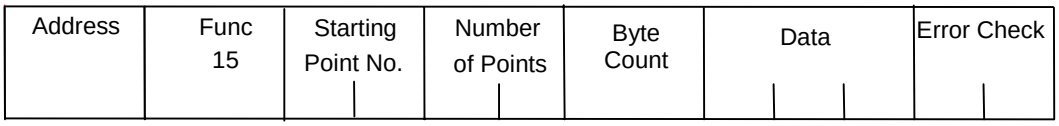
Diagnostic Force Listen-Only Mode Request (Loopback/Maintenance code 04):

- An address of 0 indicates a broadcast request. All slave stations process a broadcast re-quest.
- After receiving a Force Listen-Only mode request, the RTU device will go into the listen-only mode, will not perform a requested function, and will not send either normal or error responses to any queries. The listen-only mode is disabled when the RTU device receives an Initiate Communication Restart request or when the RTU device is powered up.
- Both bytes in the data field of a Force Listen-Only Mode request are equal to 0. The RTU device never sends a response to a Force Listen-Only Mode request.

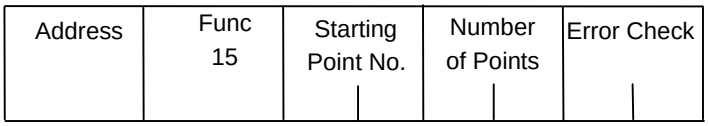
Note: Upon power up, the RTU device disables the listen-only mode and is enabled to continue sending responses to queries.

Message (15): Force Multiple Outputs

Format:



Query



Normal Response

Query:

- An address of 0 indicates a broadcast request. All slave stations process a broadcast request and no response is sent.
- The value of the function code is 15.
- The starting point number is two bytes in length and may be any value less than the highest output point number available in the attached CPU. The starting point number is equal to one less than the number of the first output point forced by this request.
- The number of points value is two bytes in length. The sum of the starting point number and the number of points value must be less than or equal to the highest output point number available in the attached CPU. The high order byte of the starting point number and number of bytes fields is sent as the first byte in each of these fields. The low order byte is the second byte in each of these fields.
- The byte count is a binary number from 1 to 256 (0 = 256). It is the number of bytes in the data field of the force multiple outputs request.
- The data field is packed data containing the values that the outputs specified by the starting point number and the number of points fields are to be forced to. Each byte in the data field contains the values that eight output points are to be forced to. The least significant bit (LSB) of the first byte contains the value that the output point whose number is equal to the starting point number plus one is to be forced to. The values for the output points are ordered by number starting with the LSB of the first byte of the data field and ending with the most significant bit (MSB) of the last byte of the data field. If the number of points is not a multiple of 8, then the last data byte contains zeros in one to seven of its highest order bits.

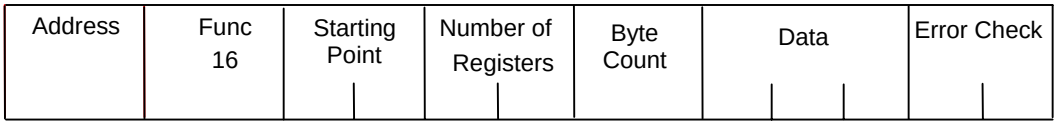
Response:

- The description of the fields in the response are covered in the query description.

Note: The force multiple outputs request is not an output override command. The outputs specified in this request are ensured to be forced to the values specified only at the beginning of one sweep of the user logic.

Message (16): Preset Multiple Registers

Format:



Query



Normal Response

Query:

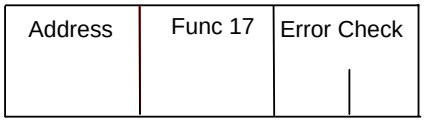
- An address of 0 indicates a broadcast request. All slave stations process a broadcast re-quest and no response is sent.
- The value of the function code is 16.
- The starting register number is two bytes in length. The starting register number may be any value less than the highest register number available in the attached CPU. It is equal to one less than the number of the first register preset by this request.
- The number of registers value is two bytes in length. It must contain a value from 1 to 125 inclusive. The sum of the starting register number and the number of registers value must be less than or equal to the highest register number available in the attached CPU. The high order byte of the starting register number and number of registers fields is sent as the first byte in each of these fields. The low order byte is the second byte in each of these fields.
- The byte count field is one byte in length. It is a binary number from 2 to 250 inclusive. It is equal to the number of bytes in the data field of the preset multiple registers request. Note that the byte count is equal to twice the value of the number of registers.
- The registers are returned in the data field in order of number with the lowest number register in the first two bytes and the highest number register in the last two bytes of the data field. The number of the first register in the data field is equal to the starting register number plus one. The high order byte is sent before the low order byte of each register.

Response:

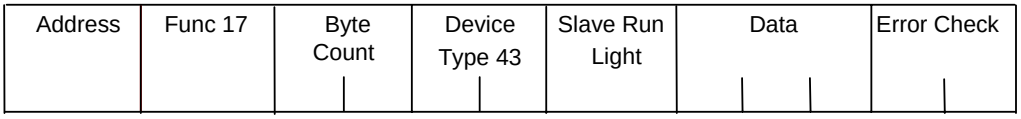
- The description of the fields in the response are covered in the query description.

Message (17): Report Device Type

Format:



Query



Normal Response

Query:

The Report Device Type query is sent by the master to a slave in order to learn what type of programmable control or other computer it is.

- An address of zero is not allowed as this cannot be a broadcast request.
- The function code is 17.

Response:

- The byte count field is one byte in length and is equal to 5.
- The device type field is one byte in length and is equal to 43 (hexadecimal) for the PACSystems
- The slave run light field is one byte in length. The slave run light byte is equal to OFFH if the CPU is in RUN mode. It is equal to 0 if the CPU is not in RUN mode.
- The data field contains three bytes. For PACSystems CPUs, the first byte is the Minor Type, and the remaining bytes are zeroes. The following table lists minor types.

Response Data (Minor Type)	***CPU Model***
02 hex	IC698CPE010
04 hex	IC698CPE020
05 hex	IC698CRE020
06 hex	IC698CPE030
08 hex	IC698CPE040
0A hex	IC695CPU310
10 hex	IC695CPU320

Message (22): Mask Write 4x Memory

Modifies the contents of a specified 4*x* register using a combination of an AND mask, an OR mask, and the register's current contents. The function can be used to set or clear individual bits in the register. Broadcast is not supported.

Query

The query specifies the 4x reference to be written, the data to be used as the AND mask, and the data to be used as the OR mask.

The function's algorithm is:

Result = (Current Contents AND And_Mask) OR (Or_Mask AND And_Mask)

For example,

	Hex	***Binary***	
Current Contents	12	0001	0010
And_Mask	F2	1111	0010
Or_Mask	25	0010	0101
And_Mask	0D	0000	1101
Result	17	0001	0111

Note: If the Or_Mask value is zero, the result is simply the logical ANDing of the current contents and And_Mask. If the And_Mask value is zero, the result is equal to the Or_Mask value.

Note: The contents of the register can be read with the Read Holding Registers function (function code 03). They could, however, be changed subsequently as the controller scans its user logic program.

Example of a Mask Write to register 5 in slave device 17, using the above mask values:

Field Name	***Example (Hex)***
Slave Address	11
Function	16
Reference Address Hi	00
Reference Address Lo	04
And_Mask Hi	00
And_Mask Lo	F2
Or_Mask Hi	00
Or_Mask Lo	25
Error Check (LRC or CRC)	--

Response

The normal response is an echo of the query. The response is returned after the register has been written.

Message (23): Read Write 4x Memory

Performs a combination of one read and one write operation in a single Modbus transaction. The function can write new contents to a group of 4*x* registers, and then return the contents of another group of 4*x* registers. Broadcast is not supported.

Query

The query specifies the starting address and quantity of registers of the group to be read. It also specifies the starting address, quantity of registers, and data for the group to be written. The byte count field specifies the quantity of bytes to follow in the write data field.

Here is an example of a query to read six registers starting at register 5, and to write three registers starting at register 16, in slave device 17:

Field Name	***Example (Hex)***
Slave address	11
Function	17
Read Reference Address Hi	00
Read Reference Address Lo	04
Quantity to Read Hi	00
Quantity to Read Lo	06
Write Reference Address Hi	00
Write Reference Address Lo	0F
Quantity to Write Hi	00
Quantity to Write Lo	03
Byte Count	06
Write Data 1 Hi	00
Write Data 1 Lo	FF
Write Data 2 Hi	00
Write Data 2 Lo	FF
Write Data 3 Hi	00
Write Data 3 Lo	FF
Error Check (LRC or CRC)	--

Response

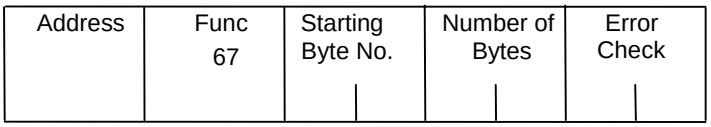
The normal response contains the data from the group of registers that were read. The byte count field specifies the quantity of bytes to follow in the read data field.

Here is an example of a response to the query:

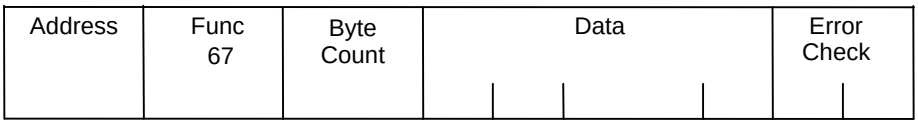
Field Name	***Example (Hex)***
Slave Address	11
Function	17
Byte Count	0C
Read Data 1 Hi	00
Read Data 1 Lo	FE
Read Data 2 Hi	0A
Read Data 2 Lo	CD
Read Data 3 Hi	00
Read Data 3 Lo	01
Read Data 4 Hi	00
Read Data 4 Lo	03
Read Data 5 Hi	00
Read Data 5 Lo	0D
Read Data 6 Hi	00
Read Data 6 Lo	FF
Error Check (LRC or CRC)	--

Message (67): Read Scratch Pad Memory

Format:



Query



Normal Response

Query:

- An address of 0 is not allowed as this cannot be a broadcast request.
- The function code is equal to 67.
- The starting byte number is two bytes in length and may be any value less than or equal to the highest scratch pad memory address available in the attached CPU as indicated in the table below. The starting byte number is equal to the address of the first scratch pad memory byte returned in the normal response to this request.
- The number of bytes value is two bytes in length. It specifies the number of scratch pad memory locations (bytes) returned in the normal response. The sum of the starting byte number and the number of bytes values must be less than two plus the highest scratch pad memory address available in the attached CPU. The high order byte of the starting byte number and number of bytes fields is sent as the first byte in each of these fields. The low order byte is the second byte in each of the fields.

Response:

- The byte count is a binary number from 1 to 256 (0 = 256). It is the number of bytes in the data field of the normal response.
- The data field contains the contents of the scratch pad memory requested by the query. The scratch pad memory bytes are sent in order of address. The contents of the scratch pad memory byte whose address is equal to the starting byte number is sent in the first byte of the data field. The contents of the scratch pad memory byte whose address is equal to one less than the sum of the starting byte number and number of bytes values is sent in the last byte of the data field.

RTU Scratch Pad

The entire scratch pad is updated every time an external READ request is received by the PACSystems RTU slave. All scratch pad locations are *read only*. The scratch pad is a byte-oriented memory type.

RTU Scratch Pad Memory Allocation

SP Address	Field Identifier	Bits							
		7	6	5	4	3	2	1	0
00	CPU Run Status	0	0	0	0	See note 1.			
01	CPU Command Status	Bit pattern same as SP(00)							
02 03	CPU Type	Major[2a] (in hexadecimal) Minor[2b] (in hexadecimal)							
04 – 0B	CPU SNP ID	7 ASCII characters + termination character (00h)							
0C 0D	CPU Firmware Revision No.	Major (in BCD) Minor (in BCD)							
0E 0F	Communications Management Module (CMM) Firmware Revision No.	Major Minor							
10—11	Reserved	00h							
12	Node Type Identifier	PACSystems 43 (hexadecimal)							
13—15	Reserved	00h							
16	RTU Station Address	1—247 (decimal)							
17	Reserved	00h							
18—33	Sizes of Memory Types[3]								
18—1B	Register Memory	%R size (words)							
1C—1F	Analog Input Table	%AI size (words)							
20—23	Analog Output Table	%AO size (words)							
24—27	Input Table	%I size (bits)							
28—2B	Output Table	%O size (bits)							
2C—2F	Internal Discrete Memory	%M size (bits)							
30—33	User Program Code	The amount of program memory occupied by the logic program.							
34—FF	Reserved	00h							

Scratch Pad Memory Allocation Notes

[1] 0000 = Run_Enabled 0100 = Halted
0001 = Run_Disabled 0101 = Suspended
0010 = Stopped 0110 = Stopped_IO_Enabled

[2a] CPU Major Type Codes:
PACSystems 0x43

[2b] PACSystems Minor Types for CPU:
see Message (17) Report Device Type

[3] Scratch Pad Bytes 18h-33h:

Four bytes hold the hexadecimal length of each memory type with the most significant word reserved for future expansion. For example, the default register memory size of 1024 words (0400h) would be returned in the following format:

Word	Least	Significant	Most	Significant
SP Byte	18	19	1A	1B
Contains	00	04	00	00

Communication Errors

Serial link communication errors are divided into three groups:

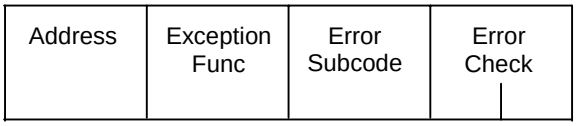
- Invalid Query Message
- Serial Link Time Outs
- Invalid Transaction

Invalid Query Message

When the communications module receives a query addressed to itself, but cannot process the query, it sends one of the following error responses:

	Subcode
Invalid Function Code	1
Invalid Address Field	2
Invalid Data Field	3
Query Processing Failure	4

The format for an error response to a query is as follows:



The address reflects the address provided on the original request. The exception function code is equal to the sum of the function code of the query plus 128. The error subcode is equal to 1, 2, 3, or 4. The value of the subcode indicates the reason the query could not be processed.

Invalid Function Code Error Response (1)

An error response with a subcode of 1 is called an invalid function code error response. This response is sent by a slave if it receives a query whose function code is not equal to 1 through 8, 15, 16, 17, or 67.

Invalid Address Error Response (2)

An error response with a subcode of 2 is called an invalid address error response. This error response is sent in the following cases:

1. The starting point number and number of points fields specify output points or input points that are not available in the attached CPU (returned for function codes 1, 2, 15).
2. The starting register number and number of registers fields specify registers that are not available in the attached CPU (returned for function codes 4, 16).
3. The starting analog input number and analog input number fields specify analog inputs that are not available in the attached CPU (returned for function code 3).
4. The point number field specifies an output point not available in the attached CPU (returned for function code 5).
5. The register number field specifies a register not available in the attached CPU (returned for function code 6).

6. The analog input number field specifies an analog input number not available in the at-attached CPU (returned for function code 3).
7. The diagnostic code is not equal to 0, 1, or 4 (returned for function code 8).
8. The starting byte number and number of bytes fields specify a scratch pad memory address that is not available in the attached CPU (returned for function code 67).

Invalid Data Value Error Response (3)

An error response with a subcode of 3 is called an invalid data value error response. This response is sent in the following cases:

The first byte of the data field is not equal to 0 or 255 (FFh) or the second byte of the data field is not equal to 0 for the Force Single Output Request (Function Code 5) or the initiate communication restart request (function code 8, diagnostic code 1). The two bytes of the data field are not both equal to 0 for the Force Listen-Only request (Function Code 8, Diagnostic Code 4). This response is also sent when the data length specified by the memory address field is longer than the data received.

Query Processing Failure Error Response (4)

An error response with a subcode of 4 is called a query processing failure response. This error response is sent by a RTU device if it properly receives a query but communication between the associated CPU and the CMM fails.

Serial Link Timeout

The only cause for a RTU device to timeout is if an interruption to a data stream of 4 character times occurs while a message is being received. If this occurs the message is considered to have terminated and no response will be sent to the master. There are certain timing considerations due to the characteristics of the slave that should be taken into account by the master. After sending a query message, the master should wait an appropriate amount of time for slave turnaround before assuming that the slave did not respond to the request. Slave turnaround time is affected by the Controller Communications Window time and the CPU sweep time, as described in “RTU Slave Turnaround Time” on page 14-24.

Invalid Transactions

If an error occurs during transmission that does not fall into the category of an invalid query message or a serial link time-out, it is known as an invalid transaction. Types of errors causing an invalid transaction include:

- Bad CRC.
- The data length specified by the memory address field is longer than the data received.
- Framing or overrun errors.
- Parity errors.

If an error in this category occurs when a message is received by the slave serial port, the slave does not return an error message; rather the slave ignores the incoming message, treating the message as though it was not intended for it.

RTU Slave/SNP Slave Operation With Programmer Attached

A port that has been configured for RTU Slave protocol can switch to SNP protocol if an SNP master such as a programmer begins communicating to the port. The programmer must use the same serial communications parameters (baud rate, parity, stop bits, etc.) as the currently active RTU Slave protocol for it to be recognized. When the CPU recognizes the SNP master, the CPU removes the RTU Slave protocol from the port and installs SNP Slave as the active protocol.

The SNP protocol that is installed in this case has the following fixed characteristics:

- The SNP ID is set to blank. Therefore the SNP master must use a blank ID in the SNP attach message. This also means that this capability is only useful for point-to-point connections.
- The turnaround time is set to 0 ms.
- The idle timeout is set to 10 seconds.

After the programmer is removed, there is a slight delay (equal to the idle timeout) before the CPU recognizes its absence. During this time, no messages are processed on the port. The CPU detects removal of the programmer as an SNP Slave protocol timeout. Therefore, it is important to be careful when disabling timeouts used by the SNP Slave protocol.

When the CPU recognizes the programmer disconnect, it reinstalls RTU Slave protocol unless a new protocol has been configured in the meantime. In that case, the CPU installs the new protocol instead.

Example

1. Port 1 is running RTU Slave protocol at 9600 baud.
2. A programmer is attached to port 1. The programmer is using 9600 baud.
3. The CPU installs SNP Slave on port 1 and the programmer communicates normally.
4. The programmer stores a new configuration to port 1. The new configuration sets the port for SNP Slave at 4800 baud (it will not take effect until the port loses communications with the programmer).
5. When the CPU loses communications with the programmer, the new configuration takes effect.

SNP Slave Protocol

PACSystems CPUs can communicate with Machine Edition software through either Port 1 or Port 2 using SNP slave protocol.

CPU port 1 is wired as an RS-232 Data Communications Equipment (DCE) port, and can be connected directly using straight-through cable to one of the serial ports of a PC running Machine Edition or other SNP master software.

CPU port 2 is wired for RS-485. If the SNP master does not have an RS-485 port, an RS-485/RS-232 converter is required. The RX3i can use converter IC690ACC901, which uses +5VDC from the serial port. The RX7i CPU port 2 does not support IC690ACC901 and requires an externally powered converter.

PACSystems provides the *break free* version of SNP, so that the SNP master does not need to issue a break signal as part of the SNP attach sequence. However, the CPU responds appropriately if a break signal is detected, by resetting the protocol to wait for another attach sequence from the master.

PACSystems supports both point-to-point connections (single master/single slave) and multi-drop connections (single master/multiple slaves).

For details on SNP protocol, refer to the *Serial Communications User's Manual,* GFK-0582.

Permanent Datagrams

Permanent datagrams survive after the SNP session that created them has been terminated. This allows an SNP master device to periodically retrieve datagram data from a number of different PLCs on a multi-drop link, without the master having to establish and write the datagram each time it reconnects to the PLC.

The maximum number of permanent datagrams that can be established is 32. When this limit is reached, additional requests to establish datagrams are denied. One or more of the permanent datagrams will need to be cancelled before others can be established. Since the permanent datagrams are not automatically deleted when the SNP session is terminated, this limit prevents an inordinate amount of these datagrams from being established.

Permanent datagrams do not survive a power-cycle.

Communication Requests (COMM_REQs) for SNP

The PACSystems serial ports 1 and 2 currently do not provide SNP Master service, nor do they support COMM_REQ functions for SNP commands. However, those COMM_REQ functions can be used with PCM/CMM modules that are configured to provide SNP service. For more information, refer to the *Serial Communications User's Manual*, GFK-0582.

Chapter 15

Fault Handling

This chapter explains the PACSystems fault handling system, provides definitions of fault extra data, and suggests corrective actions for faults.

Faults occur in the control system when certain failures or conditions happen that affect the operation and performance of the system. Some conditions, such as the loss of an I/O module or rack, may impair the ability of the PLC to control a machine or process. Other conditions may be indicated, such as when a new module comes online and becomes available for use. Some conditions, such as a failed battery, may be displayed to inform or alert the user.

Any detected fault is recorded in the controller fault table or the I/O fault table, as applicable.

Information in this chapter is organized as follows:

- Overview 15-2
- Using the Fault Tables 15-4
- System Handling of Faults 15-8
- Controller Fault Descriptions and Corrective Actions 15-14
- I/O Fault Descriptions and Corrective Actions 15-38

Overview

The PACSystems CPU detects three classes of faults:

Fault Class	*Examples*
Internal Failures (Hardware)	Non-responding modules Failed battery Memory checksum errors
External I/O Failures (Hardware)	Loss of rack or module Addition of rack or module Loss of Genius I/O block
Operational Failures	Communication failures Configuration failures Password access failures

System Response to Faults

Hardware failures require that either the system be shut down or the failure be tolerated. I/O failures may be tolerated by the control system, but they may be intolerable by the application or the process being controlled. Operational failures are normally tolerated.

Faults have three attributes:

Fault Table Affected	I/O fault table controller fault table
Fault Action	Fatal Diagnostic Informational
Configurability	Configurable Nonconfigurable

Fault Tables

The PACSystems CPU maintains two fault tables, the controller fault table for internal CPU faults and the I/O fault table for faults generated by I/O devices (including I/O controllers). For more information, see “Using the Fault Tables” on page 15-4.

Fault Actions and Fault Action Configuration

Fatal faults cause the fault to be recorded in the appropriate table, diagnostic variables to be set, and the system to be stopped. Only fatal faults cause the system to stop.

Diagnostic faults are recorded in the appropriate table, and any diagnostic variables are set. Informational faults are only recorded in the appropriate table.

Fault Action	***Response by CPU***
Fatal	Log fault in fault table. Set fault references. Go to Stop/Fault mode.
Diagnostic	Log fault in fault table. Set fault references.
Informational	Log fault in fault table.

The hardware configuration can be used to specify the fault action of some fault groups. For these groups, the fault action can be configured as either fatal or diagnostic. When a fatal or diagnostic fault within a configurable group occurs, the CPU executes the configured fault action instead of the action specified within the fault.

Note: The fault action displayed in the expanded fault details indicates the fault action specified by the fault that was logged, but not necessarily the executed fault action. To determine what action was executed for a particular fault in a configurable fault group, you must refer to the hardware configuration settings.

Faults that are part of configurable fault groups:

Fault Action Displayed in Fault Table	Informational	Diagnostic	Fatal
Fault Action Executed	Informational	Diagnostic or Fatal. Determined by action selected in Hardware Configuration.	Diagnostic or Fatal. Determined by action selected in Hardware Configuration.

Faults that are part of nonconfigurable fault groups:

Fault Action Displayed in Fault Table	Informational	Diagnostic	Fatal
Fault Action Executed	Informational	Diagnostic	Fatal

Using the Fault Tables

To display the fault tables in Logic Developer software,

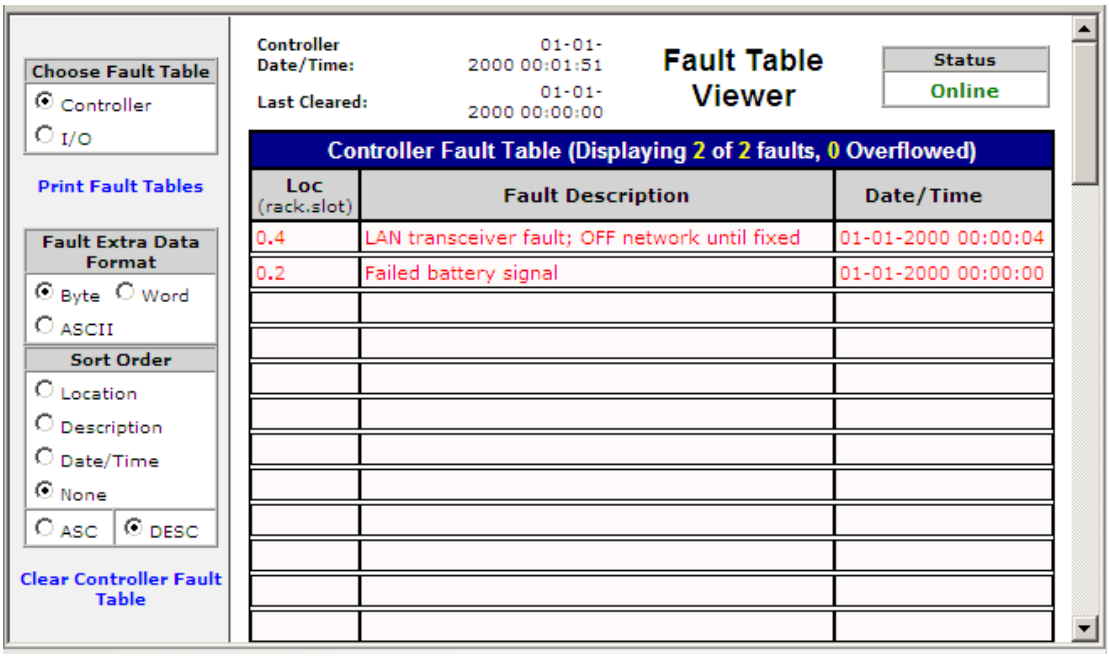
1. Go online with the PACSystems.
2. Select the Project tab in the Navigator, right click the Target node and choose Diagnostics. The Fault Table Viewer appears.

The controller fault table and the I/O fault table display the following information:

PLC Time/Date	The current date and time of the CPU.
Last Cleared	The date and time faults were last cleared from the fault table. This information is maintained by the PLC.
Status	Displays “Updating” while the programmer is reading the fault table. Status is “Online” when update is complete.
Total Faults	The total number of faults since the table was last cleared.
Entries Overflowed	The number of entries lost because the fault table has overflowed since it was cleared. Each fault table can contain up to 64 faults.

Controller Fault Table

The controller fault table displays CPU faults such as password violations, configuration mismatches, parity errors, and communications errors.



The controller fault table provides the following information for each fault:

Location Identifies the location of the fault by rack.slot.

Description Corresponds to a fault *group*, which is identified in the fault Details.

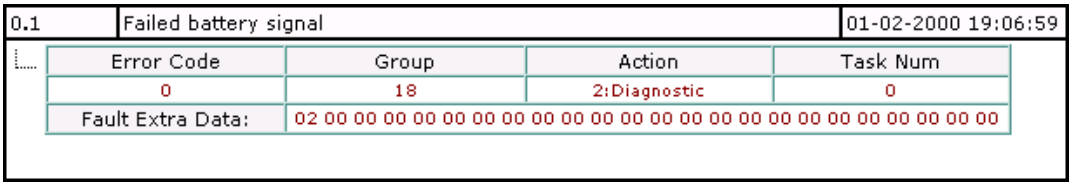
Date/Time The date and time the fault occurred based on the CPU clock.

Details To view detailed information, click the fault entry. See “Viewing Controller Fault Details” for more information.

Viewing Controller Fault Details

Note: The fault action displayed in the expanded fault details indicates the fault action specified by the fault that was logged, but not necessarily the executed fault action. To determine what action was executed for a particular fault in a configurable fault group, you must refer to the hardware configuration settings.

To see controller fault details, click the fault entry. The detailed information box for the fault appears. (To close this box, click the fault.)



The detailed information for controller faults includes the following:

Error Code Further identifies the fault. Each fault group has its own set of error codes.

Group Group is the highest classification of a fault. It identifies the general category of the fault. The fault description text displayed by your programming software is based on the fault group and the error codes.

Action Fatal, Diagnostic, or Informational. For definitions of these actions, refer to page 15-3.

Task Number Not used for most faults. When used, provides additional information for Technical Support representatives.

Fault Extra Data Provides additional information for troubleshooting by Technical Support engineers. Explanations of this information are provided as appropriate for specific faults in “Controller Fault Descriptions and Corrective Actions” on page 15-14.

User-Defined Faults

User-defined faults can be logged in the controller fault table. When a user-defined fault occurs, it is displayed in the appropriate fault table as “Application Msg (error_code):” and may be followed by a descriptive message up to 24 characters. The user can define all characters in the descriptive message. Although the message must end with the null character, e.g., zero (0), the null character does not count as one of the 24 characters. If the message contains more than 24 characters, only the first 24 characters are displayed.

Certain user-defined faults can be used to set a system status reference (%SA0081–%SA0112).

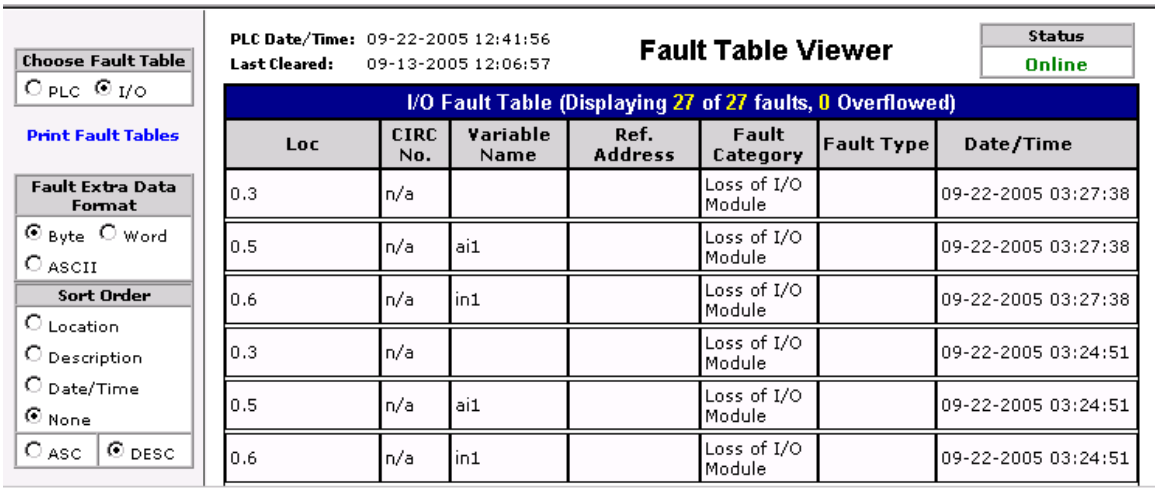
User-defined faults are created using Service Request 21, which is described in chapter 9.

Note: When a user-defined fault is displayed in the Controller Fault table, a value of -32768 (8000 hex) is added to the error code. For example, the error code 5 will be displayed as -32763.

I/O Fault Table

The I/O fault table displays I/O faults such as circuit faults, address conflicts, forced circuits, I/O module addition/loss faults and I/O bus faults.

The fault table displays a maximum of 64 faults. When the fault table is full, it displays the earliest 32 faults (33—64) and the last 32 faults (1—32). When another fault is received, fault 32 is shoved out of the table. In this way, the first 32 faults are preserved for the user to view.



The I/O fault table provides the following information for each fault:

Location Identifies the location of the fault by rack.slot location, and sometimes bus and buss address.

CIRC No. When applicable, identifies the specific I/O point on the module.

Variable Name If the fault is on a point that is mapped to an I/O variable, and the variable is set to publish (either internal or external), the I/O fault table displays the variable name. Unpublished I/O variables will not be displayed in this field.

Ref. Address* If the fault is on a point that is mapped to a reference address, this field identifies the I/O memory type and location (offset) that corresponds to the point experiencing the fault. When a Genius device fault or local analog module fault occurs, the reference address refers to the first point on the block where the fault occurred.

Fault Category Specifies a general classification of the fault.

Fault Type Consists of subcategories under certain fault categories. Set to zero when not applicable to the category.

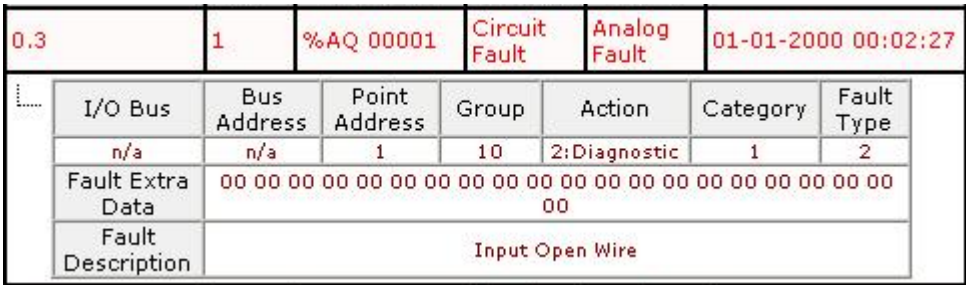
Date/Time The date and time the fault occurred based on the CPU clock.

Details To view detailed information, click the fault entry. See “Viewing I/O Fault Details” for more information.

***Note:** The Reference Address field displays 16-bits and %W memory has a 32-bit range. Addresses in %W are displayed correctly for offsets in the 16-bit range (≤65,535). For %W offsets greater than 16-bits, the I/O Fault Table displays a blank reference address.

Viewing I/O Fault Details

To see I/O fault details, click the fault entry. The detailed information box for the fault appears. (To close this box, click the fault.)



0.3	1	%AQ 00001	Circuit Fault	Analog Fault	01-01-2000 00:02:27

I/O Bus	Bus Address	Point Address	Group	Action	Category	Fault Type
n/a	n/a	1	10	2:Diagnostic	1	2
Fault Extra Data	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00					
Fault Description	Input Open Wire					

The detailed information for I/O faults includes:

I/O Bus When the module in the slot is a Genius Bus Controller (GBC), this number is always one.

Bus Address The serial bus address of the Genius device that reported or has the fault.

Point Address Identifies the point on the I/O device that has the fault when the fault is a point-type fault.

Group Fault group is the highest classification of a fault. It identifies the general category of the fault.

Action Fatal, Diagnostic, or Informational. For definitions of these actions, refer to page 15-3.

Category Identifies the category of the fault.

Fault Type Identifies the fault type by number. Set to zero when not applicable to the category.

Fault Extra Data Provides additional information for troubleshooting by Technical Support engineers. Explanations of this information are provided as appropriate for specific faults in “I/O Fault Descriptions and Corrective Actions” on page 15-36.

Fault Description Provides a specific fault code when the I/O fault category is a circuit fault (discrete circuit fault, analog circuit fault, low-level analog fault) or module fault. It is set to zero for other fault categories.

System Handling of Faults

The system fault references listed below can be used to identify the specific type of fault that has occurred. (A complete list of system status references is provided in chapter 7.)

System Fault Reference	Address	Description
#ANY_FLT	%SC0009	Any new fault in either table since the last power-up or clearing of the fault tables
#SY_FLT	%SC0010	Any new system fault in the controller fault table since the last power-up or clearing of the fault tables
#IO_FLT	%SC0011	Any new fault in the I/O fault table since the last power-up or clearing of the fault tables
#SY_PRES	%SC0012	Indicates that there is at least one entry in the controller fault table
#IO_PRES	%SC0013	Indicates that there is at least one entry in the I/O fault table
#HRD_FLT	%SC0014	Any hardware fault
#SFT_FLT	%SC0015	Any software fault

On power-up, the system fault references are cleared. If a fault occurs, the positive contact transition of any affected reference is turned on the sweep after the fault occurs. The system fault references remain on until both fault tables are cleared or All Memory in the CPU is cleared.

System Fault References

When a system fault reference is set, additional fault references are also set. These other types of faults are listed in “Fault References for Configurable Faults” below and “Fault References for Non-Configurable Faults” on page 15-10.

Fault References for Configurable Faults

Fault (Default Action)	Address	Description	May Also Be Set
#SBUS_ER (diagnostic)	%SA0032	System bus error. All system bus error faults are logged as informational.	#HRD_FLT, #SY_PRES, #SY_FLT
#SFT_IOC[1] (diagnostic)	%SA0029	Non-recoverable software error in an I/O Controller (IOC).	#IO_FLT, #IO_PRES, #SFT_FLT
#LOS_RCK[2] (diagnostic)	%SA0012	Loss of rack (BRM failure, loss of power) or missing a configured rack.	#SY_FLT, #SY_PRES, #IO_FLT, #IO_PRES
#LOS_IOC[3] (diagnostic)	%SA0013	Loss of I/O Controller or missing a configured Bus Controller.	#IO_FLT, #IO_PRES
#LOS_IOM (diagnostic)	%SA0014	Loss of I/O module (does not respond), or missing a configured I/O module.	#IO_FLT, #IO_PRES
#LOS_SIO (diagnostic)	%SA0015	Loss of intelligent module (does not respond), or missing a configured module.	#SY_FLT, #SY_PRES
#IOC_FLT (diagnostic)	%SA0022	Non-fatal bus or I/O Controller error, more than 10 bus errors in 10 seconds. (Error rate is configurable.)	#IO_FLT, #IO_PRES
#CFG_MM (fatal)	%SA0009	Configuration mismatch. Wrong module type detected. The PLC does not check the configuration parameter settings for individual modules such as Genius I/O blocks.	#SY_FLT, #SY_PRES
#OVR_TMP (diagnostic)	%SA0008	CPU temperature has exceeded its normal operating temperature.	#SY_FLT, #SY_PRES

1. The #SFT_IOC software fault will have the same action as what you set for #LOS_IOC.
2. When a Loss of Rack or Addition of Rack fault is logged, individual loss or add faults for each module in that rack are usually not generated.
3. Even if the #LOS_IOC fault is configured as Fatal, the PLC will not go to STOP/FAULT unless both GBCs of an internal redundant pair fail.

Note: If the fault action for a fault logged to the fault table is informational, the configured action is not used. For example, if the logged fault action for an SBUS_ERR is informational, but you configure it as fatal, the action is still informational.

Fault References for Non-Configurable Faults

Fault	Address	Description	Result
#PS_FLT	%SA0005	Power supply fault	Sets #SY_FLT, #SY_PRES
#HRD_CPU (fatal)	%SA0010	CPU hardware fault (such as failed memory device or failed serial port).	Sets #SY_FLT, #SY_PRES, #HRD_FLT
#HRD_SIO (diagnostic)	%SA0027	Non-fatal hardware fault on any module in the system, such as failure of a serial port on a LAN interface module.	Sets #SY_FLT, #SY_PRES, #HRD_FLT
#SFT_SIO (diagnostic)	%SA0031	Non-recoverable software error in a LAN interface module.	Sets #SY_FLT, #SY_PRES, #SFT_FLT
#PB_SUM (fatal)	%SA0001	Program or block checksum failure during power-up or in Run mode.	Sets #SY_FLT, #SY_PRES
#LOW_BAT (diagnostic)	%SA0011	Not supported for the CPU battery. For details, see “Battery Status (Group 18)” on page 15 28.	Sets #SY_FLT, #SY_PRES
#OV_SWP (diagnostic)	%SA0002	Constant sweep time exceeded.	Sets #SY_FLT, #SY_PRES
#SY_FULL #IO_FULL (diagnostic)	%SA0022	Controller fault table full (64 entries). I/O fault table full (64 entries).	Sets #SY_FLT, #SY_PRES, #IO_FLT, #IO_PRES
#APL_FLT (diagnostic)	%SA0003	Application fault.	Sets #SY_FLT, #SY_PRES
#ADD_RCK* (diagnostic)	%SA0017	New rack added, extra rack, or previously faulted rack has returned.	Sets #SY_FLT, #SY_PRES
#ADD_IOC (diagnostic)	%SA0018	Extra IOC, previously faulted I/O Controller is no longer faulted.	Sets #IO_FLT, #IO_PRES
#ADD_IOM (diagnostic)	%SA0019	Extra IO module, or previously faulted I/O module is no longer faulted.	Sets #IO_FLT, #IO_PRES
#ADD_SIO (diagnostic)	%SA0020	New intelligent module is added, or previously faulted module no longer faulted.	Sets #SY_FLT, #SY_PRES
#IOM_FLT (diagnostic)	%SA0023	Point or channel on an I/O module; a partial failure of the module.	Sets #IO_FLT, I#O_PRES
#NO_PROG (information)	%SB0009	No application program is present at power-up. Should only occur the first time the PLC is powered up or if the battery-backed RAM containing the program fails.	PLC will not go to Run mode; it continues executing Stop mode sweep until a valid program is loaded. This can be a “null” program that does nothing. Sets #SY_FLT and #SY_PRES.
#BAD_RAM (fatal)	%SB0010	Corrupted program memory at power-up. Program could not be read and/or did not pass checksum tests.	Sets #SY_FLT and #SY_PRES.
#WIND_ER (information)	%SB0001	Window completion error. Servicing of Controller Communications or Logic Window was skipped. Occurs in Constant Sweep mode.	Sets #SY_FLT and #SY_PRES.
#BAD_PWD (information)	%SB0011	Change of privilege level request to a protection level was denied; bad password.	Sets #SY_FLT and #SY_PRES.
#NUL_CFG (fatal)	%SB0012	No configuration present upon transition to Run mode. Running without a configuration is equivalent to suspending the I/O scans.	Sets #SY_FLT and #SY_PRES.

Fault	Address	Description	Result
#SFT_CPU (fatal)	%SB0013	CPU software fault. A non-recoverable error has been detected in the CPU. May be caused by Watchdog Timer expiring.	PLC immediately transitions to Stop/Halt mode. The only activity permitted is communication with the programmer. To be cleared, PLC power must be cycled. Sets SY_FLT, SY_PRES, and SFT_FLT.
#STOR_ER (fatal)	%SB0014	Download of data to PLC from the programmer failed; some data in PLC may be corrupted.	PLC will not transition to Run mode. This fault is not cleared at power-up, intervention is required to correct it. Sets SY_FLT and SY_PRES.

* When a Loss of Rack or Addition of Rack fault is logged, individual loss or add faults for each module in that rack are usually not generated.

Using Fault Contacts

Fault (-[F]-) and no-fault (-[NF]-) contacts can be used to detect the presence of various faults in the system. These contacts cannot be overridden. The following table shows the state of fault and no-fault contacts.

Condition	[F]	[NF]
Fault Present	ON	OFF
Fault Absent	OFF	ON

Fault Locating References (Rack, Slot, Bus, Module)

The PACSystems CPU supports reserved fault names for each rack, slot, bus, and module. By programming these names on the FAULT and NOFLT contact instructions, logic can be executed in response to faults associated with configured racks and modules.

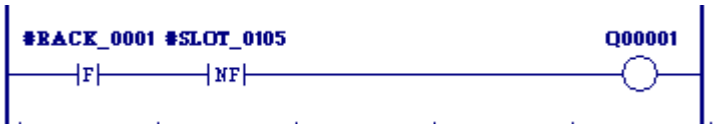
Fault Locating Reference Name Format

These fault names can only be programmed on the FAULT and NOFLT contacts. The reserved fault names are always available. It is not necessary to enable a special option, such as point faults.

Fault Reference Type	Reserved Name	Comment
Rack	#RACK_000r	Where r is rack number 0 to 7.
Slot	#SLOT_0rss	Where r is rack number 0 to 7 and ss is slot number 0 to 17.
Bus	#BUS_0rssb (Genius only)	Where r is rack number 0 to 7, ss is slot number 0 to 17, and b is the Genius bus number.
Module	#M_rssbmmm (Genius only)	Where r is rack number 0 to 7, ss is slot number 0 to 17, b is the Genius bus number, and mmm is the Serial Bus Address (SBA) number 000 to 031.

These fault names do not correspond to %SA, %SB, %SC, or to any other reference type. They are mapped to a memory area that is not user-accessible. Only the name is displayed.

Fault Reference Name Examples:



#RACK_0001 represents rack 1.

#SLOT_0105 represents rack 1, slot 5.

#BUS_02041 represents rack 2, slot 4, bus 1.

#M_2061028 represents rack 2, slot 6, bus 1, Genius module 28.

Note: When a slot level failure fault is reported to the fault tables, all bus and module fault locating references associated with that slot are set (the FAULT contact passes power flow, and the NOFLT contact does not pass power flow), regardless of what type of module it is. Conversely, when a slot level reset fault is reported to the fault tables, all bus and module fault locating references are cleared (the FAULT contact does not pass power flow, and the NOFLT contact passes power flow).

Behavior of Fault Locating References

At power-up, all fault locating references are cleared in the PLC. When a fault is logged, the PLC transitions the state of the affected reference(s). The state of the fault reference remains in the fault state until one of the following actions occurs:

- Both the PLC and the I/O fault tables are cleared through your programming software either by clearing each table individually or clearing the entire PLC memory.
- The associated device (rack, I/O module, or Genius device) is added back into the system. Whenever an “Addition of. . . ” fault is logged, the PLC initializes all fault references associated with the device to the NoFlt state. These references remain in the NoFlt state until another fault associated with the device is reported. (This could take several seconds for distributed I/O faults, especially if the bus controller has been reset.)

Note: These fault references are set for informational purposes only. They should not be used to qualify I/O data. The I/O point fault references (described on page 15-13) may be used to qualify I/O data. The PLC does not halt execution as a result of setting a fault locating reference to the Fault state.

The fault references have a cascading effect. If there is a problem in the module located at rack 5, slot 6, bus 1, module 29, the following fault references are set: RACK_05, SLOT_0506, BUS_05061, and M_5061029. There will only be one entry in the fault table to describe the problem with the module. The fault table does not show separate entries pertaining to the rack, slot, and bus in this case.

If an analog base module (IC697ALG230) is lost, the fault locating reference for that module is set. The fault locating references for its expander modules (IC697ALG440 and ALG441) are not set as a result of the loss. Therefore, any fault locating references to an expander module should also reference the base module to verify that the module or its base have not been lost.

Using Point Faults

Point faults pertain to external I/O faults, although they are also set due to the failure of associated higher-level internal hardware (for example, IOC failure or loss of a rack). To use point faults, they must be enabled in Hardware Configuration on the Memory parameters tab of the CPU.

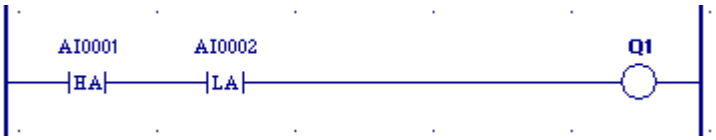
When enabled, a bit for each discrete I/O point and a byte for each analog I/O channel are allocated in PLC memory. The PLC memory used for point faults is included in the total reference table memory size. The FAULT and NOFLT contacts described in “Using Fault Contacts” on page 15-11 provide access to the point faults.

The full support of point fault contacts depends on the capability of the I/O module. Some Series 90-30 modules do not support point fault contacts. The point fault contacts for these modules remain all off, unless a Loss of I/O Module occurs, in which case the RX3i CPU turns on all point fault contacts associated with the lost module.

Using Alarm Contacts

High (-[HA]-) and low (-[LA]-) alarm contacts are used to represent the state of the analog input module comparator function. To use alarm contacts, point faults must first be enabled in Hardware Configuration on the Memory parameters tab of the CPU.

The following example logic uses both high and low alarm contacts.



Note: HA and LA contacts do not create an entry in a fault table.

Controller Fault Descriptions and Corrective Actions

Each fault explanation contains a fault description and instructions to correct the fault. Many fault descriptions have multiple causes. In these cases, the error code and additional fault information are used to distinguish among fault conditions sharing the same fault description.

Controller Fault Groups

Group	Name	Default Fault Action*	Configurable
1	Loss of or Missing Rack	Diagnostic	Yes
4	Loss of or Missing Option Module	Diagnostic	Yes
5	Addition of, or Extra Rack	N/A	No
8	Reset of, Addition of, or Extra Option Module	N/A	No
11	System Configuration Mismatch	Fatal**	Yes
12	System Bus Error	Fatal	Yes
13	CPU Hardware Failure	N/A	No
14	Module Hardware Failure	N/A	No
16	Option Module Software Failure	N/A	No
17	Program or Block Checksum Failure Group	N/A	No
18	Battery Status Group	N/A	No
19	Constant Sweep Time Exceeded	N/A	No
20	System Fault Table Full	N/A	No
21	I/O Fault Table Full	N/A	No
22	User Application Fault	N/A	No
24	CPU Over Temperature	Diagnostic	Yes
128	System Bus Failure	N/A	No
129	No User Program on Power-up	N/A	No
130	Corrupted User Program on Power-up	N/A	No
131	Window Completion Failure	N/A	No
132	Password Access Failure	N/A	No
134	Null System Configuration for Run Mode	N/A	No
135	CPU System Software Failure	N/A	No
137	Communications Failure During Store	N/A	No
140	Non-critical CPU Software Event	N/A	No

* The fault action indicated is not applicable if the fault is displayed as informational. Faults displayed as informational, always behave as informational.

** If a system configuration mismatch occurs when the CPU is in Run mode, the fault action will be Diagnostic regardless of the fault configuration. For additional information, see “Fault Parameters” in chapter 3.

Loss of or Missing Rack (Group 1)

The fault group Loss of or Missing Rack occurs when the system cannot communicate with an expansion rack because the BTM (Bus Transmitter Module) in the main rack failed, the BRM (Bus Receiver Module) in the expansion rack failed, power failed in the expansion rack, or the expansion rack was configured in the configuration file but did not respond during power-up.

Default action: Diagnostic. Configurable.

1, Rack Lost

The PLC generates this error when the main rack can no longer communicate with an expansion rack. The error is generated for each expansion rack that exists in the system.

Correction

(1) Power off the system. Verify that both the BTM and the BRM are seated properly in their respective racks and that all cables are properly connected and seated.

(2) Replace the cables.

(3) Replace the BRM.

(4) Replace the BTM.

2, Rack Not Responding

The PLC generates this error when the configuration file stored from the programmer indicates that a particular expansion rack should be in the system but none responds for that rack number.

Correction

(1) Check rack number jumper behind power supply—first on missing rack and then on all other racks—for duplicated rack numbers.

(2) Update the configuration file if a rack should not be present.

(3) Add the rack to the hardware configuration if a rack should be present and one is not.

(4) Power off the system. Verify that both the BTM and the BRM are seated properly in their respective racks and that all cables are properly connected and seated.

(5) Replace the cables.

(6) Replace the BRM.

(7) Replace the BTM.

(8) Check for Termination Plug on last BRM.

Loss of or Missing Option Module (Group 4)

The fault group Loss of or Missing Option Module occurs when a LAN interface module, BTM, or BRM fails to respond. The failure may occur at power-up or store of configuration if the module is missing or during operation if the module fails to respond. This may also occur due to hot removal of an option module.

Default action: Diagnostic. Configurable

3C hex/60 decimal, Module in Firmware Update Mode

The PLC generates this error when it finds a module in Firmware Update mode. Modules in this mode will not communicate with the CPU.

Correction

(1) Run the firmware update utility for the module.

(2) Reset the module with the push-button.

(3 Power-cycle the entire system.

(4 Power-cycle the rack containing the module.

63 hex/99 decimal, Module Hot Removed

The PLC logs this fault when it detects hot removal of an option module such as the LAN interface module. No correction necessary

All Others, Module Failure During Configuration

The PLC generates this error when a module fails during power-up or configuration store.

Correction

(1) Power off the system. Replace the module located in that rack and slot.

(2) If the board is located in an expansion rack, verify BTM/BRM cable connections are tight and the modules are seated properly; verify the addressing of the expansion rack.

(3) Replace the BTM.

(4) Replace the BRM.

(5) Replace the rack.

Addition of, or Extra Rack (Group 5)

This fault group occurs when a configured expansion rack with which the CPU could not communicate comes online or is powered on, or an unconfigured rack is found.

Action: Nonconfigurable.

1, Extra Rack

Correction

(1) Check rack jumper behind power supply for correct setting.

(2) Update the configuration file to include the expansion rack.

Note: No correction necessary if rack was just powered on.

Reset of, Addition of, or Extra Option Module (Group 8)

The fault group Reset of, Addition of, or Extra Option Module occurs when an option module (LAN interface module, BTM, etc.) comes online, is reset, is hot inserted or a module is found in the rack but is not configured.

Action: Nonconfigurable.

8, LAN Interface Restart Complete, Running Utility

The LAN Interface module has restarted and is running a utility program.

Correction

Refer to the LAN Interface manual, GFK-0868 or GFK-0869 (previously GFK-0533).

7, Extra Option Module

Correction

(1) Update the configuration file to include the module.

(2) Remove the module from the system.

E Hex/14 Decimal, Option Module Hot inserted

The PLC logs this fault when it detects hot insertion of an option module such as the LAN interface module. No correction necessary

Note: When configuration is cleared or stored, a reset fault is generated for every intelligent option module physically present in the system.

System Configuration Mismatch (Group 11)

The fault group Configuration Mismatch occurs when the module occupying a slot is different from that specified in the configuration file. When the GBC generates the mismatch because of a Genius block, the second byte in the Fault Extra Data field contains the bus address of the mismatched block.

Default action: Fatal. Configurable.

Note: If a system configuration mismatch occurs when the CPU is in Run mode, the fault action will be Diagnostic regardless of the fault configuration. For additional information, see "Fault Parameters" in chapter 3.

2, Genius I/O Block Model Number Mismatch

The PLC generates this fault when the configured and physical Genius I/O blocks have different model numbers.

Correction

(1) Replace the Genius I/O block with one corresponding to the configured module.

(2) Update the configuration file.

Fault Extra Data for Genius I/O Block Model Number Mismatch

Byte	Value
[0]	FF (flag byte)
[1]	Serial Bus address
[2]	Installed module type (See "Installed/Configured Module Types" on page 15-18.)
[3]	Configured module type (See "Installed/Configured Module Types" on page 15-18.)

Installed/Configured Module Types (Bytes 2 and 3 of Fault Extra Data)

Number		Description
Decimal	Hexadecimal	
4	4	Genius Network Interface (GENI)
5	5	Phase B Hand Held Monitor
6	6	Phase B Series Six GBC with Diagnostics
7	7	Phase B Series Six GBC without Diagnostics
8	8	PLCM/Series Six
9	9	PLCM/Series 90-40
10	A	Series 90-70 Single Channel Bus Controller
11	B	Series 90-70 Dual Channel Bus Controller
12	C	Series 90-10 Genius Communications Module
13	D	Series 90-30 Genius Communications Module
32	20	High Speed Counter
69	45	Phase B 115Vac 8-point (2 amp) Grouped Block
70	46	Phase B 115Vac/125Vdc 8-point Isolated Block
70	46	Phase B 115Vac/125Vdc 8-point Isolated Block without Failed Switch
71	47	Phase B 220Vac 8-point Grouped Block
72	48	Phase B 24-48Vdc 16-point Proximity Sink Block

Number		Description
Decimal	Hexadecimal	
72	48	Phase B 24Vdc 16-point Proximity Sink Block
73	49	Phase B 24-48Vdc 16-point Source Block
73	49	Phase B 24Vdc 16-point Proximity Source Block
74	4A	Phase B 12-24Vdc 32-point Sink Block
75	4B	Phase B 12-24Vdc 32-point Source Block
76	4C	Phase B 12-24Vdc 32-point 5V Logic Block
77	4D	Phase B 115Vac 16-point Quad State Input Block
78	4E	Phase B 12-24Vdc 16-point Quad State Input Block
79	4F	Phase B 115/230Vac 16-point Normally Open Relay Block
80	50	Phase B 115/230Vac 16-point Normally Closed Relay Block
81	51	Phase B 115Vac 16-point AC Input Block
82	52	Phase B 115Vac 8-point Low-Leakage Grouped Block
127	7F*	Genius Network Adapter (GENA)
131	83	Phase B 115Vac 4-input, 2-output Analog Block
132	84	Phase B 24Vdc 4-input, 2-output Analog Block
133	85	Phase B 220Vac 4-input, 2-output Analog Block
134	86	Phase B 115Vac Thermocouple Input Block
135	87	Phase B 24Vdc Thermocouple Input Block
136	88	Phase B 115Vac RTD Input Block
137	89	Phase B 24/48Vdc RTD Input Block
138	8A	Phase B 115Vac Strain Gauge/mV Analog Input Block
139	8B	Phase B 24Vdc Strain Gauge/mV Analog Input Block
140	8C	Phase B 115Vac 4-input, 2-output Current Source Analog Block
141	8D	Phase B 24Vdc 4-input, 2-output Current Source Analog Block

**GENA Application ID Numbers*

If the model number is 7F hex (Genius Network Adapter), the block may be one of the following. (The GENA Application ID is shown for reference.)

Number		
Decimal	Hexadecimal	Description
131	83	115Vac/230Vac/125Vdc Power Monitor Module
132	84	24/48Vdc Power Monitor Module
160	A0	Genius Remote 90-70 Rack Controller

4, I/O Type Mismatch

The PLC generates this fault when the physical and configured I/O types of Genius grouped blocks are different.

Correction

(1) Remove the indicated Genius module and install the module indicated in the configuration file.

(2) Update the Genius module descriptions in the configuration file to agree with what is physically installed.

Fault Extra Data for I/O Type Mismatch

Byte	***Value***
[0]	FF
[1]	Bus address
[2]	Installed module's I/O type
[3]	Configured module's I/O type

Genius Installed Module I/O Types (Byte 2 of Fault Extra Data)

Value	***Description***
01	Input only
02	Output only
03	Combination

Genius Configured Module I/O Types (Byte 3 of Fault Extra Data)

Value		***Description***
Decimal	***Hexadecimal***	
0	0	Discrete input
1	1	Discrete output
2	2	Analog input
3	3	Analog output
4	4	Discrete grouped
5	5	Analog grouped
20	14	Analog in, discrete in
21	15	Analog in, discrete out
24	18	Analog in, discrete grouped
30	1E	Analog out, discrete in
31	1F	Analog out, discrete out
34	22	Analog out, discrete grouped
50	32	Analog grouped, discrete in
51	33	Analog grouped, discrete out
54	36	Analog grouped, discrete grouped

8, Analog Expander Mismatch

The CPU generates this error when the configured and physical Analog Expander modules have different model numbers.

Correction

(1) Replace the Analog Expander module with one corresponding to configured module.

(2) Update the configuration file.

9, Genius I/O Block Size Mismatch

The CPU generates this error when block configuration size does not match the configured size.

Correction

Reconfigure the block.

Fault Extra Data for Genius I/O Block Size Mismatch

Byte	Value
[0]	FF
[1]	Bus address
[2]	Module's broadcast data length
[3]	Configured module's broadcast data length

A hex/10 decimal, Unsupported Feature

Configured feature not supported by this revision of the module.

Correction

(1) Update the module to a revision that supports the feature.

(2) Change the module configuration.

Fault Extra Data for Unsupported Feature

Byte	Value
[8]	Contains a reason code indicating what feature is not supported. 0x5 – GBC revision too old 0x6 – Only supported in main rack

E hex/14 decimal, LAN Duplicate MAC Address

This LAN Interface module has the same MAC address as another device on the LAN. The module is off the network.

Correction

(1) Change the module's MAC address.

(2) Change the other device's MAC address.

F hex/15 decimal, LAN Duplicate MAC Address Resolved

Previous duplicate MAC address has been resolved. The module is back on the network. This is an informational message. No correction required.

10 hex/16 decimal, LAN MAC Address Mismatch

MAC address programmed by softswitch utility does not match configuration stored from software.

Correction

Change MAC address on softswitch utility or in software.

11 hex/17 decimal, LAN Softswitch/Modem mismatch

Configuration of LAN module does not match modem type or configuration programmed by softswitch utility.

Correction

(1) Correct configuration of modem type.

(2) Consult LAN Interface manual for configuration setup.

19 hex/25 decimal, DCD Length Mismatch

Correction

See Fault Extra Data.

Fault Extra Data for DCD Length Mismatch

Byte	***Value***
[0]	FF
[1]	Bus address
[2]	Module's directed data length
[3]	Configured module's directed data length

25 hex/37 decimal, Controller Reference Out of Range

A reference on either the trigger, disable, or I/O specification is out of the configured limits.

Correction

Modify the incorrect reference to be within range, or increase the configured size of the reference data.

26 hex/38 decimal, Bad Program Specification

The I/O specification of a program is corrupted.

Correction

Contact GE Fanuc Field Service.

27 hex/39 decimal, Unresolved or Disabled Interrupt Reference

The CPU generates this error when an interrupt trigger reference is either out of range or disabled in the I/O module's configuration.

Correction

(1) Remove or correct the interrupt trigger reference.

(2) Update the configuration file to enable this particular interrupt.

43 hex/67 decimal, Module Configuration Failure

Module configuration was not successfully accepted by the module.

Correction

Check fault table for other module-specific faults for possible reasons why the module did not accept the configuration. Check that the configuration for the module is correct and valid.

4C hex/76 decimal, ECC jumper is enabled, but should be disabled

If the CPU firmware does not support redundancy, the ECC jumper must be in the disabled position.

Correction

Set the ECC jumper to the disabled position (jumper on one pin or removed entirely).

All Others, Module and Configuration do not Match

The CPU generates this fault when the module occupying a slot is not of the same type that the configuration file indicates.

Correction

(1) Replace the module in the slot with the type indicated in the configuration file.

(2) Update the configuration file.

System Bus Error (Group 12)

The fault group System Bus Error occurs when the CPU encounters a bus error.

Default action: Diagnostic. Configurable.

4, Unrecognized VME Interrupt Source

The CPU generates this error when a module generates an interrupt not expected by the CPU (unconfigured or unrecognized).

Correction

Ensure that all modules configured for interrupts have corresponding interrupt declarations in the program logic.

CPU Hardware Failure (Group 13)

The fault group CPU Hardware occurs when the CPU detects a hardware failure, such as a RAM failure or a communications port failure.

When a CPU Hardware failure occurs, the OK LED will flash on and off to indicate that the failure was not serious enough to prevent Controller Communications to retrieve the fault information.

Action: Nonconfigurable.

6E hex/110 decimal, Time-of-Day Clock not Battery-Backed

The battery-backed value of the time-of-day clock has been lost.

Correction

(1) Replace the battery. Do not remove power from the main rack until replacement is complete. Reset the time-of-day clock using your programming software.

(2) Replace the module.

0A8 hex/168 decimal, Critical Overtemperature Failure

CPU's critical operating temperature exceeded.

All Others

Correction

Replace the module.

Fault Extra Data for CPU Hardware Failure

For a RAM failure in the CPU (one of the faults reported as a CPU hardware failure), the address of the failure is stored in the first four bytes of the field.

Module Hardware Failure (Group 14)

The fault group Module Hardware Failure occurs when the CPU detects a non-fatal hardware failure on any module in the system, for example, a serial port failure on a LAN interface module. The fault action for this group is Diagnostic.

Action: Nonconfigurable.

1A0 hex/416 decimal, Missing 12 Volt Power Supply

A power supply that supplies 12 volts is required to operate the LAN Interface module.

Correction

(1) Install/replace a GE Fanuc 100 watt power supply.

(2) Connect an external VME power supply that supplies 12 volts.

1C2 - 1C6 hex (450 – 454 decimal), LAN Interface Hardware Failure

Refer to the LAN Interface manual, GFK-0868 or GFK-0869 (previously GFK-0533), for a description of these errors.

All Others, Module Hardware Failure

A module hardware failure has been detected.

Correction

Replace the affected module.

Option Module Software Failure (Group 16)

The fault group Option Module Software Failure occurs when:

- A non-recoverable software failure occurs on an intelligent option module.
- The identification data read from a module indicates that the module is a GE Fanuc module but the module type is not a supported GE Fanuc type.
- The Ethernet Interface logs an event in its Ethernet exception log.

Action: Nonconfigurable.

1, Unsupported Board Type

The CPU generates this fault when the identification data read from a board indicates that the board is a GE Fanuc board but the type of board is not one of the GE Fanuc board types.

Correction

(1) Upload the configuration file and verify that the software recognizes the board type in the file. If there is an error, correct it, download the corrected configuration file, and retry.

(2) Display the controller fault table on the programmer. Contact GE Fanuc Field Service, giving them all the information contained in the fault entry.

2, 3, COMMREQ Frequency Too High

COMM_REQs are being sent to a module faster than it can process them.

Correction

Change the application program to send COMM_REQs to the affected module at a slower rate or monitor the completion status of each COMMREQ before sending the next.

4, More Than One BTM in a Rack

There is more than one BTM present in the rack.

Correction

Remove one of the BTMs from the rack; there can only be one in a CPU rack.

>4, Option Module Software Failure

Software failure detected on an option module.

Correction

(1) Reload software into the indicated module.

(2) Replace the module.

>4, LAN System Software Fault

The Ethernet interface software has detected an unusual condition and recorded an event in its exception log. The Fault Extra Data contains the corresponding event in the Ethernet exception log, which may also be viewed by the Ethernet Interface's Station Manager function. The first two digits of Fault Extra Data contain the two-digit Event type; the remaining data correspond to the four-digit values for Entry 2 through Entry 6. Some exceptions may also contain optional multi-byte SCode and other data.

Correction

For information on interpreting the fault extra data, refer to the PACSystems TCP/IP Communications Station Manager Manual, GFK-2225, Appendix B.

Program or Block Checksum Failure (Group 17)

The fault group Program or Block Checksum Failure occurs when the CPU detects error conditions in program or blocks received by the PLC. It also occurs during Run mode background checking. In all cases, the Fault Extra Data field of the controller fault table record contains the name of the program or block in which the error occurred.

Action: Nonconfigurable.

All Error Codes, Program or Block Checksum Failure

The CPU generates this error when a program or block is corrupted.

Correction

(1) Clear CPU memory and retry the store.

(2) Examine C application for errors.

(3) Display the controller fault table on the programmer. Contact GE Fanuc Field Service, giving them all the information contained in the fault entry.

Fault Extra Data for Program or Block Checksum Failure

The name of the offending program block is contained in the first eight bytes of the Fault Extra Data field.

Battery Status (Group 18)

Faults in this group occur when the CPU detects a failed battery, or when a module such as a LAN interface module reports a low or failed battery condition.

Action: Nonconfigurable.

0, Failed Battery Signal

The battery in the CPU module (or other module that has a battery) has failed or is disconnected.

Note: This fault is not supported by the CPU310. For more information, refer to the *Important Product Information* document, GFK-2329.

Correction

Replace the battery. Do not remove power from the rack until replacement is complete.

1, Low Battery Signal

This fault is not supported for CPU low battery status.

This fault may be logged when an I/O or special-purpose module has reported a low battery.

When a Failed Battery Signal is logged, this fault is also logged.

Correction

Replace the battery. Do not remove power from the rack until replacement is complete.

Constant Sweep Time Exceeded (Group 19)

The fault group Constant Sweep Exceeded occurs when the CPU operates in Constant Sweep mode and detects that the sweep has exceeded the constant sweep timer. The fault extra data contains the name of the folder in eight bytes.

Action: Nonconfigurable.

0, Constant Sweep

Correction

If Constant Sweep (0):

(1) Increase constant sweep time.

(2) Remove logic from application program.

Note: Error code 1 is not used.

System Fault Table Full (Group 20)

The fault group System Fault Table Full occurs when the Controller Fault Table reaches its limit (see page 15-4).

Action: Nonconfigurable.

0, System Fault Table Full

Correction

Clear the controller fault table.

I/O Fault Table Full (Group 21)

The fault group I/O Fault Table Full occurs when the I/O Fault Table reaches its maximum configured limit (see page 15-6). To avoid loss of additional faults, clear the earliest entry from the table.

Action: Nonconfigurable.

0, I/O Fault Table Full

Correction

Clear the I/O fault table.

User Application Fault (Group 22)

The fault group Application Fault occurs when the CPU detects a fault in the user program.

Action: Nonconfigurable.

2, Software Watchdog Timer Expired

The CPU generates this error when the watchdog timer expires. The CPU stops executing the user program and enters Stop/Halt mode. To recover, cycle power to the CPU with battery disconnected. Causes of timer expiration include: Looping, via jump, very long program, etc.

Correction

(1) Determine what caused the expiration (logic execution, external event, etc.) and correct.

(2) Use the system service function block to restart the watchdog timer.

7, Application Stack Overflow

Block call depth has exceeded the CPU capability.

Correction

Increase the program's stack size or adjust application program to reduce nesting.

11 hex/17 decimal, Program Run Time Error

A run-time error occurred during execution of a program.

Correction

Correct the specific problem in the application.

22 hex/34 decimal, Unsupported Protocol

Hardware does not support configured protocol.

33 hex/51 decimal, Flash Read Failed

Possible causes:

(1) Files not in flash. (May be caused by power cycle during flash write.)

(2) Could not read from flash because OEM protection is enabled.

34 hex/52 decimal, Memory Reference Out of Range

A user logic memory reference, computed during logic execution, is out of range. Includes indirect references, array element references, and potentially other types of references.

Correction

Correct logic or adjust memory size in hardware configuration.

35 hex/53 decimal, Divide by zero attempted in user logic.

User logic contained a divide by zero operation.

Correction

Correct logic.

36 hex/54 decimal, Operand is not byte aligned.

A variable in user logic is not properly byte-aligned for the requested operation.

Correction

Correct logic or adjust memory size in hardware configuration.

3B hex /59 decimal, PSB called by a block whose %L or %P memory is not large enough to accommodate this reference.

Parameterized blocks do not have their own %L data, but instead inherit the %L data of their calling blocks. If %L references are used within a parameterized block and the block is called by _MAIN, %L references are inherited from the %P references wherever encountered in the parameterized block (for example, %L0005 = %P0005). For a discussion of the use of local data with parameterized blocks, refer to "Parameterized Blocks and Local Data" in chapter 6.

Correction

Determine which block called the parameterized subroutine block and increase the size of %L or %P memory allocated to the calling block. (To do this, change the Extra Local Words setting in the block's Properties.)

The maximum size of %L or %P is 8192 words per block. If your application needs more space, consider changing some %P or %L references to %R, %W, %AI, or %AQ. These changes require a recompilation of the program block and a Stop Mode store to the CPU.

It is possible, by using Online Editing in the programming software to cause a parameterized block to use %L higher than allowed because of the way it inherits data. To correct this condition, delete the %L variables from the logic and then remove the unused variables from the variable list. These changes require a recompilation of the program block and a Stop Mode store to the CPU.

CPU Over Temperature (Group 24)

Default action: Diagnostic. Configurable.

1, Overtemperature failure.

CPU's normal operating temperature exceeded.

Correction

Turn off CPU to allow heat to disperse and install a fan kit to regulate temperature.

Power Supply Fault (Group 25)

Action: Nonconfigurable.

1, Power supply failure.

Unknown power supply failure.

Correction

Replace power supply module.

2, Power supply overloaded

The load on the power supply has reached its rated maximum

Correction

Replace power supply with a higher capacity model or reconfigure system to reduce load on power supply.

3, Power supply switched off

The switch on the power supply was moved to the OFF position.

4, Power-supply has exceeded normal operating temperature

The temperature of the power supply is a just a few degrees from causing it to turn off.

Correction

Turn off system to allow heat to disperse. Install a fan kit to regulate temperature.

No User Program on Power-Up (Group 129)

The fault group No User Program on Power-Up occurs when the CPU powers up with its memory preserved but no user program exists in the PLC. The CPU detects the absence of a user program on power-up; the controller stays in Stop mode.

Action: Nonconfigurable.

Correction

Download an application program before attempting to go to Run mode.

Corrupted User Program on Power-Up (Group 130)

The fault group Corrupted User Program on Power-Up occurs when the CPU detects corrupted user RAM. The CPU will remain in Stop mode.

Action: Nonconfigurable.

1, Corrupted user RAM on power-up

The CPU generates this error when it detects corrupted user RAM on power-up.

Correction

(1) Cycle power without battery.

(2) Examine any C applications for errors.

(3) Replace the battery on the CPU.

(4) Replace the expansion memory board on the CPU.

(5) Replace the CPU.

7, User memory not preserved over power cycle

The CPU generates this error when it detects a battery failure that occurred while the PLC was powered down.

If this fault occurs on a power cycle when the battery was not detached or replaced, the battery has failed and should be replaced.

Correction

Replace the battery on the CPU.

Window Completion Failure (Group 131)

The fault group Window Completion Failure is generated by the pre-logic and end-of-sweep processing software in the PLC. The fault extra data contains the name of the task that was executing when the error occurred.

Action: Nonconfigurable.

0, Window Completion Failure

The CPU generates this error when it is operating in Constant Sweep mode and the constant sweep time was exceeded before the programmer window had a chance to begin executing.

Correction

Increase the constant sweep timer value.

1, Logic Window Skipped

The logic window was skipped due to lack of time to execute.

Correction

(1) Increase base cycle time.

(2) Reduce Communications Window time.

Password Access Failure (Group 132)

The fault group Password Actual Failure occurs when the CPU receives a request to change to a new privilege level and the password included with the request is not valid for that level.

Action: Nonconfigurable.

0, Password Access Failure

Correction

Retry the request with the correct password.

Null System Configuration for Run Mode (Group 134)

The fault group Null System Configuration for Run Mode occurs when the CPU transitions from Stop to one of the Run modes and a configuration file is not present. The transition to Run is permitted, but no I/O scans occur.

Action: Informational. Nonconfigurable.

0, Null System Configuration for Run Mode

Correction

Download a configuration file.

CPU System Software Failure (Group 135)

Faults in this group are generated by the operating software of the CPU. They occur at many different points of system operation. When a fatal fault occurs, the CPU immediately transitions to Stop/Halt. The only activity permitted when the CPU is in this mode is communications with the programmer. The only method of clearing this condition is to cycle power on the PLC with the battery disconnected.

Action: Nonconfigurable.

5A hex/90 decimal, User Shut Down Requested

The CPU generates this informational alarm when SVCREQ #13 (User Shut Down) executes in the application program.

Correction

None required. Information-only alarm.

94 hex/148 decimal, Units Contain Mismatched Firmware, Update Recommended

This fault is logged each time the redundancy state changes and the redundant CPUs contain incompatible firmware.

Correction

Ensure that redundant CPUs have compatible firmware.

DA hex/218 decimal, Critical Overtemperature Failure

CPU's critical operating temperature exceeded.

Correction

Turn off CPU to allow heat to disperse and install a fan kit to regulate temperature.

All Others, CPU Internal System Error

An internal system error has occurred that should **not** occur in a production system.

Correction

Display the controller fault table on the programmer. Contact GE Fanuc Field Service and give them all the information contained in the fault entry.

Communications Failure During Store (Group 137)

This fault group occurs during the store of programs or blocks and other data to the PLC. The stream of commands and data for storing programs or blocks and data starts with a special start-of-sequence command and terminates with an end-of-sequence command. This fault is logged if communications with the programming device performing the store is interrupted or any other failure that terminates the store occurs. As long as this fault is present in the system, the controller will not transition to Run mode. This fault is *not* automatically cleared on power-up; you must specifically clear the condition.

Action: Nonconfigurable.

0, Communications Failure During Store

Correction

Clear the fault and retry the download of the program or configuration file.

1, Communications Lost During Run Mode Store

Communications or power was lost during a Run Mode Store. The new program or block was not activated and was deleted.

Correction

Perform the Run Mode Store again. This fault is diagnostic.

2, Communications Lost During Cleanup for Run Mode Store

Communications was lost, or power was lost during the cleanup of old programs or blocks during a Run Mode Store. The new program or block is installed, and the remaining programs and blocks were cleaned up.

Correction

None required. This fault is informational.

3, Power Lost During a Run Mode Store

Power was lost in the middle of a Run Mode Store.

Correction

Delete and restore the program. This error is fatal.

Noncritical CPU Software Event (Group 140)

This group is used for recording conditions in the system that may provide valuable information to Technical Support.

Default action: Nonconfigurable.

Error Codes	***Descriptions***
1-30	Events during power-up
31-50	Events on the serial port or in a serial protocol
51 and greater	Miscellaneous internal system events

Correction

No corrective action is required unless this fault occurs with other specific faults. The fault may contain useful information for Technical Support if other problems are encountered.

I/O Fault Descriptions and Corrective Actions

The I/O fault table reports the following data about faults:

- Fault Group
- Fault Action
- Fault category
- Fault type
- Fault description

All faults have a fault category, but a fault type and fault group may not be listed for every fault. To view the detailed information pertaining to a fault, click the fault entry in the I/O Fault Table.

Note: The model number mismatch and I/O type mismatch faults are reported in the controller fault table under the System Configuration Mismatch group. They are not reported in the I/O fault table.

Fault Extra Data

An I/O fault table entry contains up to 21 bytes of I/O fault extra data that contains additional information related to the fault. Not all entries contain I/O fault extra data.

I/O Fault Groups

Group Number	Group Name	Default Fault Action*	Configurable
2	Loss of or Missing IOC	Diagnostic	Yes
3	Loss of or Missing I/O module	Diagnostic	Yes
6	Addition or Reset of, or Extra IOC	N/A	No
7	Addition of or Extra I/O module	N/A	No
9	IOC or I/O Bus Fault	Diagnostic	Yes
10	I/O Module Fault	N/A	No
15	IOC Software Failure	Same As Group 2 **	Yes
16	Module Software Failure	N/A	No
133	Genius Block Address Mismatch	N/A	No

* The fault action indicated is not applicable if the fault is displayed as informational. Faults displayed as informational, always behave as informational.

** The fault action for the IOC Software Failure group 15 always matches the action used by the Loss of or Missing IOC group 2. If the Loss of or Missing IOC group is configured, the IOC Software Failure group is also configured to take the same fault action.

I/O Fault Categories

Category	Fault Type	Fault Description	Fault Extra Data
Circuit Fault (1)	Discrete Fault (1)	Loss of User Side Power (01 hex)	Circuit Configuration
		Short Circuit in User Wiring (02 hex)	Circuit Configuration
		Sustained Overcurrent (04 hex)	Circuit Configuration
		Low or No Current Flow (08 hex)	Circuit Configuration
		Switch Temperature Too High (10 hex)	Circuit Configuration
		Switch Failure (20 hex)	Circuit Configuration
		Point Fault (83 hex)	Circuit Configuration
		Output Fuse Blown (84 hex)	Circuit Configuration
	Analog Fault (2)	Input Channel Low Alarm (01 hex)	Circuit Configuration
		Input Channel High Alarm (02 hex)	Circuit Configuration
		Input Channel Under Range (04 hex)	Circuit Configuration
		Input Channel Over Range (08 hex)	Circuit Configuration
		Input Channel Open Wire (10 hex)	Circuit Configuration
		Over Range or Open Wire (18 hex)	Circuit Configuration
		Output Channel Under Range (20 hex)	Circuit Configuration
		Output Channel Over Range (40 hex)	Circuit Configuration
		Expansion Channel Not Responding (80 hex)	Circuit Configuration
		Invalid Data (81 hex)	Circuit Configuration
	Low-Level Analog Fault (4)	Input Channel Low Alarm (01 hex)	Circuit Configuration
		Input Channel High Alarm (02 hex)	Circuit Configuration
		Input Channel Under Range (04 hex)	Circuit Configuration
		Input Channel Over Range (08 hex)	Circuit Configuration
		Input Channel Open Wire (10 hex)	Circuit Configuration
		Wiring Error (20 hex)	Circuit Configuration
		Internal Fault (40 hex)	Circuit Configuration
		Input Channel Shorted (80 hex)	Circuit Configuration
		Invalid Data (81 hex)	Circuit Configuration
	GENA (Genius Network Adapter) Fault (3)	GENA Circuit Fault (80 hex)	Byte 2:GENA Fault
Loss of Block (2)	Not Specified (0) A/D Communications Lost (1)	NA	Block Configuration Number of Input Circuits Number of Output Circuits
Addition of Block (3)	NA	NA	Block Configuration Number of Input Circuits Number of Output Circuits
I/O Bus Fault (6)	Bus Fault (1) Bus Outputs Disabled (2) SBA Conflict (3)	NA	NA
Genius Module Fault (8)	Headend Fault (0) A to D Comm. Fault (1) User Scaling Error (5) Store Fail (6)	Configuration Memory Failure (08 hex) Calibration Memory Failure (20 hex) Shared RAM Failure (40 hex) Internal Circuit Fault (80 hex) Watchdog Timeout (81 hex) Output Fuse Blown (84 hex)	NA

Category	*Fault Type*	*Fault Description*	*Fault Extra Data*
Addition of IOC (9)	NA	Extra Module (01 hex) Reset Request (02 hex)	NA
Loss of IOC (10)	NA	NA	Timeout Unexpected State Unexpected Mail Status VME Bus Error
IOC Software Fault (11)	NA	NA	NA
Forced Circuit (12)	NA	NA	Block Configuration Discrete/Analog Indication*
Unforced Circuit (13)	NA	NA	Block Configuration Discrete/Analog Indication*
Loss of I/O Module (14)	NA	NA	NA
Addition of I/O Module (15)	NA	VME Module Reset Requested (30 hex)	NA
Extra I/O Module (16)	NA	NA	NA
Extra Block (17)	NA	NA	NA
IOC Hardware Failure (18)	NA	NA	NA
GBC stopped reporting faults because too many faults have occurred (19)	GBC detected high error count on Genius Bus and dropped off the bus for at least 1.5 seconds. (1)	NA	NA
GBC Software Exception (21)	Datagram queue full (1) R/W request queue full (2) Low priority mail rejected (3) Background message received before CPU completed initialization (4) Genius software version too old (5) Excessive use of internal GBC memory (6)	NA	
Block Switch (22) – redundant Genius block switched bus	NA	NA	Block Configuration Number of Input Circuits Number of Output Circuits Rack/Slot address of GBC from which block was removed.
Block not active on redundant bus (23)	NA	NA	NA
Reset of IOC (27)	NA	NA	NA

Circuit Faults (Category 1)

Circuit faults apply to Genius I/O modules and the IC697VRD008 RTD/Strain Bridge modules. Fault extra data is available for all faults in this category. More than one condition may be present in a particular reporting of the fault.

Action: Diagnostic.

Fault Extra Data for Circuit Faults

Genius Bus Controller

Circuit fault entries use one or two bytes of the fault extra data area. If the GBC reports the fault, the first byte is generated by the GBC and the second byte contains the circuit configuration and is encoded as shown in the following table.

Value (Byte 2)	Description
1	Circuit is an input.
2	Circuit is an input.
3	Circuit is an output.

If the fault type is a GENA fault, the second byte contains the data that was reported from the GENA module in fault byte 2 of its “Report Fault” message.

VRD001 RTD/Strain Bridge

Circuit fault entries 13 bytes of the fault extra data area. The fault extra data is encoded as shown in the following table.

Bytes	Description
1--10	Used by technical support.
11	Line number
12	Module number
13	Used by technical support.

Fault Descriptions for Discrete Faults

1, Loss of User Side Power

The GBC generates this error when there is a power loss on the field wiring side of a Genius I/O block.

Correction

(1) (Only valid for Isolated I/O blocks.) Initiate “Pulse Test” COMREQ #1. Pulse test may be enabled or disabled at I/O block.

(2) Correct the power failure.

2, Short Circuit in User Wiring

The GBC generates this error when it detects a short circuit in the user wiring of a Genius block. A short circuit is defined as a current level greater than 20 amps.

Correction

Fix the cause of the short circuit.

4, Sustained Overcurrent

The GBC generates this error when it detects a sustained current level greater than 2 amps in the user wiring.

Correction

Fix the cause of the over current.

8, Low or No Current Flow

The GBC generates this error when there is very low or no current flow in the user circuit.

Correction

Fix the cause of the condition.

10 hex, Switch Temperature Too High

The GBC generates this error when the Genius block reports a high temperature in the Genius Smart Switch.

Correction

(1) Ensure that the block is installed to provide adequate circulation.

(2) Decrease the ambient temperature surrounding the block.

(3) Install RC Snubbers on inductive loads.

20 hex, Switch Failure

The GBC generates this error when the Genius block reports a failure in the Genius Smart Switch.

Correction

(1) Check for shunts across Genius output (pushbuttons).

(2) Replace the Genius I/O block.

83 hex, Point Fault

The CPU generates this error when it detects a failure of a single I/O point on a Genius I/O module.

Correction

Replace the Genius I/O block.

84 hex, Output Fuse Blown

The CPU generates this error when it detects a blown fuse on a Genius I/O output block.

Correction

(1) Determine and repair the cause of the fuse blowing; replace the fuse.

(2) Replace the block.

Fault Descriptions for Analog Faults

1, Input Channel Low Alarm

The GBC generates this error when the Genius Analog module reports a low alarm on an input channel.

Correction

Correct the condition causing the low alarm.

2, Input Channel High Alarm

The GBC generates this error when the Genius Analog module reports a high alarm on an input channel.

Correction

Correct the condition causing the high alarm.

4, Input Channel Under Range

The GBC generates this error when the Genius Analog module reports an under-range condition on an input channel.

Correction

Correct the problem causing the condition.

8, Input Channel Over Range

The GBC generates this error when the Genius Analog module reports an over-range condition on an input channel.

Correction

Correct the problem causing the condition.

10 hex/16 decimal, Input Channel Open Wire

The GBC generates this error when a Genius Analog module detects an open wire condition on an input channel.

Correction

Correct the problem causing the condition.

18 hex/24 decimal, Over Range or Open Wire

Inputs open or inputs off-scale.

Correction

Correct the problem causing the condition.

20 hex/32 decimal, Output Channel Under Range

The GBC generates this error when the Genius Analog module reports an under-range condition on an output channel.

Correction

Correct the problem causing the condition.

40 hex/64 decimal, Output Channel Over Range

The GBC generates this error when the Genius Analog module reports an over-range condition on an output channel.

Correction

Correct the problem causing the condition.

80 hex/128 decimal, Expansion Channel Not Responding

The CPU generates this error when data from an expansion channel on a multiplexed analog input board is not responding.

Correction

(1) Check wiring to the module.

(2) Replace the module.

81 hex/129 decimal, Invalid Data

The GBC generates this error when it detects invalid data from a Genius Analog input block.

Correction

Correct the problem causing the condition.

Low-Level Analog Faults

1, Input Channel Low Alarm

The GBC generates this error when the Genius Analog module reports a low alarm on an input channel.

Correction

Correct the condition causing the low alarm.

2, Input Channel High Alarm

The GBC generates this error when the Genius Analog module reports a high alarm on an input channel.

Correction

Correct the condition causing the high alarm.

4, Input Channel Under Range

The GBC generates this error when the Genius Analog module reports an under-range condition on an input channel.

Correction

Correct the problem causing the condition.

8, Input Channel Over Range

The GBC generates this error when the Genius Analog module reports an over-range condition on an input channel.

Correction

Correct the problem causing the condition.

10 hex, Input Channel Open Wire

The GBC generates this error when the Genius Analog module detects an open wire condition on an input channel.

Correction

Correct the problem causing the condition.

20 hex/32 decimal, Wiring Error

The GBC generates this error when the Genius Analog module detects an improper RTD connections or thermocouple reverse junction fault.

Correction

Correct the problem causing the condition.

40 hex/64 decimal, Internal Fault

The GBC generates this error when the Genius Analog module reports a cold junction sensor fault on a thermocouple block or an internal error in an RTD block.

Correction

Correct the problem causing the condition.

80 hex/128 decimal, Input Channel Shorted

The GBC generates this error when it detects an input channel shorted on a Genius RTD or Strain Gauge Block.

Correction

Correct the problem causing the condition.

81 hex/129 decimal, Invalid Data

The GBC generates this error when it detects invalid data from a Genius Analog input block.

Correction

Correct the problem causing the condition.

GENA Fault

The GENA Fault has no fault descriptions associated with it. GENA Fault Byte 2 is the first byte of the fault extra data.

80 hex/128 decimal

The Genius I/O operating software generates this error when it detects a failure in a GENA block attached to the Genius I/O bus.

Correction

Replace the GENA block.

Loss of Block (Category 2)

The fault category Loss of Block applies to Genius devices.

Action: Diagnostic.

Loss of Block

The GBC generates this error when it is unable to communicate to the Genius device.

Correction

(1) Verify power and wiring to the block.

(2) Replace the block.

Loss of Block - A/D Communications Fault

The GBC generates this error when it detects a failure of A/D communications on a Genius device.

Correction

(1) Verify power and serial bus wiring to the block.

(2) Replace the block.

Fault Extra Data for Loss of Block

The Loss of Block fault provides four bytes of fault extra data. The second byte contains the block configuration and is encoded as shown in the following table. The third byte specifies the number of input circuits possibly used, and the fourth byte specifies the number of output circuits possibly used.

Block Configuration (Byte 2)

Value	Description
1	Block is configured for inputs only.
2	Block is configured for outputs only.
3	Block is configured for inputs and outputs (grouped block).

Addition of Block (Category 3)

The fault category Addition of Block applies only to Genius devices. There are no fault types or fault descriptions associated with this category.

The Genius operating software generates this error when it detects that a Genius block that stopped communicating with the controller starts communicating again.

Action: Diagnostic.

Correction

Informational only. None required.

Fault Extra Data for Addition of Block

The Addition of Block fault provides four bytes of fault extra data. The second byte contains the block configuration and is encoded as shown in the following table. The third byte specifies the number of input circuits possibly used, and the fourth byte specifies the number of output circuits possibly used.

Block Configuration (Byte 2)

Value	***Description***
1	Block is configured for inputs only.
2	Block is configured for outputs only.
3	Block is configured for inputs and outputs (grouped block).

I/O Bus Fault (Category 6)

The fault category I/O Bus Faults has three fault types associated with it.

Default action: Diagnostic. Configurable.

Bus Fault

The GBC operating software generates this error when it detects a failure with a Genius I/O bus. (Generated when Error Rate in the GBC configuration is exceeded—the default Error Rate is 10 errors in a 10 second period).

Correction

(1) Determine the reason for the bus failure and correct it.

(2) Replace the GBC.

The Error Rate can be set higher than the default value if needed, but the bus should be examined electrically—use an oscilloscope for waveform check.

Bus Outputs Disabled

The GBC operating software generates this error when it times out waiting for the CPU to perform an output scan.

Correction

(1) Reduce time between GBC output scans by assigning them to scan set 1.

(2) Increase CPU software watchdog timer setting

(3) Replace the CPU.

(4) Display the controller fault table on the programmer. Contact GE Fanuc Technical Support, giving them all the information contained in the fault entry.

SBA Conflict

The GBC detected a conflict between its serial bus address and that of another device on the bus.

Correction

Adjust one of the conflicting serial bus addresses.

Module Fault (Category 8)

The fault category Module Fault has one fault type, headend fault, and eight fault descriptions. This fault category does not provide fault extra data. The default fault action for this category is Diagnostic.

08 hex, Configuration Memory Failure

The GBC generates this error when it detects a failure in a Genius block's EEPROM or NVRAM.

Correction

Replace the Genius block's electronics module.

20 hex/32 decimal, Calibration Memory Failure

The GBC generates this error when it detects a failure in a Genius block's calibration memory.

Correction

Replace the Genius block's electronics module.

40 hex/64 decimal, Shared RAM Fault

The GBC generates this error when it detects an error in a Genius block's shared RAM.

Correction

Replace the Genius block's electronics module.

80 hex/128 decimal, Module Fault

An internal failure has been detected in a module.

Correction

Replace the affected module.

81 hex/129 decimal, Watchdog Timeout

The CPU generates this error when it detects that an input module watchdog timer has expired.

Correction

Replace the input module.

84 hex/132 decimal, Output Fuse Blown

The CPU generates this error when it detects a blown fuse on an output module.

Correction

(1) Determine and repair the cause of the fuse blowing, and replace the fuse.

(2) Replace the module.

Addition of IOC (Category 9)

The fault category Addition of I/O Controller has no fault types or fault descriptions associated with it. The default fault action for this category is Diagnostic.

Addition of IOC

The CPU generates this error when an IOC that has been faulted returns to operation or when an IOC is found in the system and the configuration file indicates that no IOC is to be in that slot or when an IOC is hot inserted.

Correction

(1) No action is necessary if the faulted module is in a remote rack and is returning due to a remote rack power cycle.

(2) Update the configuration file or remove the module.

01 hex, Extra Module

Module present, but not configured.

Correction

Update the configuration file or remove the module.

02 hex, Reset Request

Module added back after reset request. No corrective action is necessary.

Loss of or Missing IOC (Category 10)

The fault category Loss of IOC has no fault types or fault descriptions associated with it.

Default action: Diagnostic. Configurable.

Note: This fault is always displayed as Fatal in the I/O Fault Table, regardless of its configured action.

The CPU generates this error when it cannot communicate with an I/O Controller and an entry for the IOC exists in the configuration file.

This fault is also logged when an IOC is hot removed (No corrective action necessary in this case).

Correction

(1) Verify that the module in the slot/bus address is the correct module.

(2) Review the configuration file and verify that it is correct.

(3) Replace the module.

(4) Display the controller fault table on the programmer. Contact GE Fanuc Field Service, giving them all the information contained in the fault entry.

Fault Extra Data for Loss of or Missing IOC

Fault extra data for Loss of or Missing IOC provides additional information for troubleshooting by Technical Support.

IOC (I/O Controller) Software Fault (Category 11)

The fault category IOC Software Fault applies to any type of I/O Controller.

Action: Fatal.

Datagram Queue Full, Read/Write Queue Full

Too many datagrams or read/write requests have been sent to the GBC.

Correction

Adjust the system to reduce the request rate to the GBC.

Response Lost

The GBC is unable to respond to a received datagram or read/write request.

Correction

Adjust the system to reduce the request rate to the GBC.

Forced and Unforced Circuit (Categories 12 and 13)

The fault categories Forced Circuit and Unforced Circuit report point conditions and therefore are not technically faults. They have no fault types or fault descriptions. These reports occur when a Genius I/O point was forced or unforced with the Hand-Held Monitor.

Action: Informational.

Fault Extra Data for Forced/Unforced Circuit

Three bytes of fault extra data are present when a circuit force is added or removed

Byte Number	***Description***	***Value***	***Description***
1	Circuit Configuration	1	Circuit is an input.
		2	Circuit is an input..
		3	Circuit is an output.
2	Analog/Discrete Information	1	Block is a discrete block.
		2	Block is an analog block.
		3	Block has both discrete and analog.

Loss of or Missing I/O Module (Category 14)

The fault category Loss of I/O Module applies to discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category.

Default action: Diagnostic. Configurable.

The CPU generates this error when it detects that an I/O module is no longer responding to commands from the CPU, or when the configuration file indicates an I/O module is to occupy a slot and no module exists in the slot. This fault is also logged when an I/O module is hot removed (No corrective action necessary in this case).

Correction

(1) Replace the module.

(2) Correct the configuration file.

(3) Display the I/O fault table on the programmer. Contact GE Fanuc Field Service, giving them all the information contained in the fault entry.

Addition of I/O Module (Category 15)

The fault category Addition of I/O Module applies to discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category.

Action: Diagnostic.

Addition of I/O Module

The CPU generates this error when an I/O module that had been faulted returns to operation or is hot inserted.

Correction

(1) No action necessary if module was removed or replaced or if the remote rack was power cycled.

(2) Update the configuration file or remove the module.

30 hex/48 decimal, VME Reset on Request

Reset of VME module was requested. No corrective action necessary.

Extra I/O Module (Category 16)

The fault category Extra I/O Module applies to discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category.

Action: Diagnostic.

The CPU generates this error when it detects an I/O module in a slot that the configuration file indicates should be empty.

Correction

(1) Remove the module. (It may be in the wrong slot.)

(2) Update and restore the configuration file to include the extra module.

Extra Block (Category 17)

The fault category Extra Block applies only to Genius I/O devices. There are no fault types or fault descriptions associated with this category.

Action: Diagnostic.

The GBC generates this error when it detects a Genius device on the bus at a serial bus address where the configuration file does not have a block.

Correction

(1) Remove or reconfigure the block. (It may be at the wrong serial bus address.)

(2) Update and restore the configuration file to include the extra block.

IOC Hardware Failure (Category 18)

The fault category IOC Hardware Failure has no fault types or fault descriptions.

Action: Diagnostic.

The Genius operating software generates this error when it detects a hardware failure in the bus communication hardware or a baud rate mismatch.

Correction

(1) Verify that the baud rate set in the configuration file for the GBC agrees with the baud rate programmed in every block on the bus.

(2) Change the configuration file and restore it, if necessary.

(3) Replace the GBC.

(4) Selectively remove each block from the bus until the offending block is isolated then replace it.

GBC Stopped Reporting Faults (Category 19)

GBC detected a high error count on the Genius I/O bus and dropped off the bus for at least 1.5 seconds.

Correction

Check for incorrect wiring, interference from other equipment, a loose connection, or a failed device on the Genius bus.

GBC Software Exception (Category 21)

1, Incoming datagram queue full

Too many datagrams or read/write requests have been sent to the GBC.

Correction

Adjust the system to reduce the request rate to the GBC.

2, Read/write request queue full

The queue for Read/Write requests in the GBC is full. The requests may be from the Genius Bus or from COMM_REQs.

Correction

Adjust the system to reduce the request rate to the GBC.

3, Low priority mail queue from GBC to CPU full

The response to the CPU was lost.

4, Genius background message requiring CPU action received before CPU completed initialization

Message was ignored.

5, GBC software version too old

Correction

Update GBC firmware.

6, Excessive use of internal GBC memory

Correction

Verify COMMREQ usage.

Block Switch (Category 22)

The Block Switch fault category has no fault types or fault descriptions.

Action: Diagnostic.

The GBC generates this error when a Genius block on redundant Genius buses switches from one bus to another.

Correction

(1) No action is required to keep the block operating.

(2) The bus that the block switched from may need to be repaired.

(a) Verify the bus wiring.

(b) Replace the I/O controller.

(c) Replace the Bus Switching Module (BSM).

Fault Extra Data for Block Switch

Byte Number	***Description***	***Value***	***Description***
1	Circuit configuration	1	Circuit is an input.
		2	Circuit is an input.
		3	Circuit is an output.
2	Block configuration	1	Block is configured for inputs only.
		2	Block is configured for outputs only.
		3	Block is configured for inputs and outputs (grouped block).
3	Number of input circuits used		
4	Number of output circuits used		

Reset of IOC (Category 27)

The fault category Reset of I/O Controller has no fault types or fault descriptions associated with it. The default fault action for this category is Diagnostic.

The CPU generates this message when an I/O Controller is reset. No corrective action necessary.

Appendix A Performance Data

This appendix contains instruction and overhead timing collected for each PACSystems CPU module. This timing information can be used to predict CPU sweep times. The information in this appendix is organized as follows:

Boolean Execution Times A-1

Instruction Timing A-2

Overhead Sweep Impact Times A-12

Note: All performance data for all CPUs was not available at the time this manual was printed. Additional information will be added to a future edition.

Boolean Execution Times

CPU Model	***Boolean execution speed per 1000 Boolean contacts/coils, typical***
IC695CPU310	0.181ms
IC695CPU320	0.047ms
IC698CPE010	0.183ms
IC698CPE020	0.078ms
IC698CRE020	0.14ms
IC698CPE030/CRE030	0.069ms
IC698CPE040/CRE040	0.02ms

Instruction Timing

The tables in this section list the execution and incremental times in microseconds for each function supported by the PACSystems CPUs. These figures were obtained by testing the following CPU versions:

Model *All functions except as listed below*	***Firmware Version***
IC695CPU310	5.0
IC695CPU320	5.10
IC698CPE010	3.6
IC698CPE020	3.5
IC698CRE020	2.04 (with ECC enabled)
IC698CPE030/CRE030	3.6
IC698CPE040/CRE040	3.6

All Models LREAL Functions, EXP_REAL, EXPT_REAL and SCAN_SET_IO	5.5

Function Execution Times

Two execution times are shown for each function.

Execution Time	Description
Enabled	Time, in microseconds, required to execute the function or function block when power flows into the function with valid inputs.
Disabled	Time, in microseconds, required to execute the function when it is not enabled.

Notes:

- All times represent typical execution time. Times may vary with input and error conditions.
- Enabled time is for single length units of word-oriented memory.
- COMMREQ time was measured between CPU and Ethernet module with NOWAIT option.
- DOIO time was measured using a discrete output module.
- Timers are updated each time they are encountered in the logic by the amount of time consumed by the last sweep.
- Performance for the CRE020 was measured with Error Checking and Correction (ECC) enabled.

	CPU310		CPU320		CPE010		CPE020		CRE020		CPE030 CRE030		CPE040 CRE040	
Function	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**
Bit Operation														
AND_WORD	3.5	1.5	0.8	0.3	3.9	1.6	1.6	0.7	2.2	0.8	1.5	0.4	0.4	0.2
AND_DWORD	3.5	1.5	0.8	0.3	3.8	1.5	1.6	0.7	2.1	0.8	1.5	0.4	0.5	0.2
OR_WORD	3.5	1.5	0.7	0.3	3.7	1.6	1.6	0.7	2.1	0.8	1.5	0.4	0.5	0.2
OR_DWORD	3.7	1.5	0.8	0.3	3.7	1.6	1.5	0.7	2.2	0.8	1.5	0.4	0.5	0.2
XOR_WORD	3.4	1.5	0.8	0.3	3.7	1.6	1.5	0.6	2.2	0.8	1.5	0.4	0.5	0.2
XOR_DWORD	3.5	1.5	0.8	0.3	3.6	1.5	1.6	0.6	2.1	0.8	1.5	0.4	0.5	0.2
NOT_WORD	2.8	1.3	0.6	0.2	2.9	1.3	1.2	0.6	1.7	0.6	1.1	0.3	0.4	0.1
NOT_DWORD	2.9	1.3	0.6	0.2	2.8	1.2	1.2	0.5	1.7	0.6	1.2	0.3	0.4	0.1
MCMP_WORD	5.5	2.2	1.4	0.6	5.8	2.1	2.5	0.9	4.0	1.1	2.4	0.7	0.8	0.3
MCMP_DWORD	5.5	2.2	1.4	0.6	5.8	2.2	2.6	0.9	4.1	1.1	2.4	0.7	0.8	0.3
SHL_WORD	4.6	2.4	1.1	0.5	4.7	2.4	2.0	1.0	2.9	1.2	1.5	0.7	0.7	0.3
SHL_DWORD	4.8	2.4	1.1	0.5	4.7	2.4	2.1	1.0	2.9	1.2	1.5	0.8	0.7	0.3
SHR_WORD	4.6	2.5	1.1	0.5	4.7	2.4	2.0	1.0	2.9	1.2	1.5	0.9	0.7	0.3
SHR_DWORD	4.6	2.4	1.1	0.5	4.7	2.4	2.0	1.0	2.9	1.2	1.5	0.9	0.7	0.3
ROL_WORD	3.3	1.5	0.6	0.3	3.0	1.5	1.3	0.7	1.8	0.7	1.3	0.3	0.3	0.2
ROL_DWORD	2.9	1.5	0.6	0.3	3.0	1.5	1.3	0.6	1.9	0.8	1.3	0.3	0.4	0.2
ROR_WORD	3.0	1.5	0.6	0.3	3.1	1.5	1.3	0.7	1.9	0.8	1.3	0.3	0.4	0.2
ROR_DWORD	3.2	1.4	0.6	0.3	2.9	1.5	1.3	0.6	2.0	0.7	1.3	0.3	0.4	0.2
BTST_WORD	3.2	1.2	0.7	0.3	3.1	1.3	1.4	0.5	2.0	0.7	1.2	0.3	0.4	0.1
BTST_DWORD	3.1	1.2	0.7	0.3	3.2	1.2	1.4	0.5	2.0	0.7	1.2	0.3	0.4	0.1
BSET_WORD	2.4	1.2	0.6	0.3	2.4	1.2	1.0	5.0	1.5	0.6	1.1	0.3	0.3	0.1
BSET_DWORD	2.4	1.1	0.6	0.3	2.3	1.1	1.0	0.5	1.5	0.6	1.1	0.3	0.3	0.1
BCLR_WORD	2.5	1.2	0.6	0.3	2.5	1.1	1.1	0.5	1.5	0.6	1.1	0.3	0.3	0.1
BCLR_DWORD	2.5	1.2	0.6	0.3	2.5	1.2	1.1	0.5	1.5	0.6	1.1	0.3	0.3	0.1
BPOS_WORD	3.9	1.3	0.9	0.3	4.0	1.4	1.7	0.6	2.4	0.7	1.6	0.3	0.6	0.2
BPOS_DWORD	4.6	1.3	1.1	0.3	4.7	1.3	2.0	0.6	2.7	0.7	1.9	0.3	0.6	0.1
Relational														
CMP_INT	3.4	1.1	0.9	0.3	3.8	1.1	1.6	0.5	2.3	0.6	1.7	0.5	0.6	0.2
CMP_DINT	3.4	1.1	0.9	0.3	3.6	1.0	1.5	0.5	2.3	0.6	1.7	0.5	0.6	0.2
CMP_UINT	3.5	1.0	0.9	0.3	3.6	1.0	1.6	0.4	1.6-	0.4	1.7	0.5	0.6	0.2
CMP_REAL	3.5	1.1	0.9	0.3	3.6	1.0	1.5	0.5	2.3	0.6	1.7	0.5	0.6	0.2
CMP_LREAL	3.9	1.2	1.1	0.4	4.2	1.5	1.8	0.7	2.1	0.5	1.8	0.8	0.6	0.1
EQ_INT	2.6	1.1	0.6	0.3	2.5	1.0	1.1	0.4	1.8	0.6	1.1	0.3	0.5	0.1
EQ_DINT	2.3	1.0	0.6	0.2	2.4	1.0	1.1	0.5	1.6	0.6	1.0	0.3	0.4	0.1
EQ_UINT	2.4	1.0	0.6	0.3	2.5	1.0	1.0	0.4	-	-	1.0	0.3	0.4	0.1
EQ_REAL	2.6	0.9	0.6	0.2	2.4	1.0	1.1	0.4	1.6	0.5	1.0	0.3	0.4	0.1
EQ_LREAL	2.9	1.1	0.9	0.3	2.9	1.4	1.2	0.6	1.4	0.5	1.3	0.6s	0.4	0.1
NE_INT	2.3	1.1	0.6	0.2	2.5	1.0	1.0	0.4	1.7	0.5	1.1	0.3	0.4	0.1
NE_DINT	2.4	1.0	0.6	0.2	2.5	0.9	1.1	0.4	1.6	0.5	1.2	0.3	0.4	0.1
NE_UINT	2.4	1.0	0.8	0.4	2.4	1.0	1.1	0.4	1.1	0.4	1.2	0.3	0.4	0.1
NE_REAL	2.5	1.0	0.7	0.3	2.5	1.0	1.1	0.4	1.6	0.5	1.2	0.3	0.4	0.1
NE_LREAL	2.9	1.2	0.8	0.3	3.0	1.5	1.3	0.6	1.5	0.5	1.3	0.6	0.4	0.1
GT_INT	2.5	1.0	0.6	0.2	2.5	1.0	1.1	0.4	1.8	0.6	1.3	0.3	0.4	0.1
GT_DINT	2.3	1.0	0.6	0.2	2.5	1.0	1.0	0.4	1.6	0.5	1.2	0.3	0.4	0.1

	CPU310		CPU320		CPE010		CPE020		CRE020		CPE030 CRE030		CPE040 CRE040	
Function	Enabled	Disabled	Enabled	Disabled	Enabled	Disabled	Enabled	Disabled	Enabled	Disabled	Enabled	Disabled	Enabled	Disabled
GT_UINT	2.4	1.0	0.6	0.2	2.4	1.0	1.1	0.4	1.1	0.4	1.2	0.3	0.4	0.1
GT_REAL	2.5	1.0	0.6	0.2	2.5	1.0	1.1	0.4	1.6	0.5	1.2	0.3	0.4	0.1
GT_LREAL	2.8	1.1	0.8	0.3	2.9	1.4	1.2	0.6	1.4	0.5	1.3	0.6	0.4	0.1
GE_INT	2.5	1.1	0.6	0.3	2.5	1.3	1.1	0.4	1.8	0.5	1.1	0.3	0.5	0.1
GE_DINT	2.3	1.0	0.6	0.2	2.4	1.2	1.1	0.5	1.6	0.6	0.9	0.3	0.4	0.1
GE_UINT	2.4	1.1	0.6	0.2	2.4	1.0	1.1	0.5	1.1	0.5	0.9	0.3	0.4	0.1
GE_REAL	2.6	1.0	0.6	0.2	2.4	1.2	1.1	0.4	1.6	0.5	1.0	0.3	0.4	0.1
GE_LREAL	2.8	1.1	0.9	0.3	2.9	1.4	1.3	0.6	1.4	0.5	1.3	0.6	0.4	0.1
LT_INT	2.5	1.0	0.6	0.2	2.5	1.0	1.1	0.4	1.8	0.5	1.3	0.3	0.4	0.1
LT_DINT	2.3	1.1	0.6	0.2	2.5	1.0	1.1	0.4	1.6	0.5	1.2	0.3	0.4	0.2
LT_UINT	2.4	1.0	0.6	0.2	2.5	1.0	1.1	0.4	1.1	0.4	1.2	0.3	0.4	0.2
LT_REAL	2.5	1.0	0.7	0.3	2.5	1.0	1.1	0.4	1.6	0.6	1.2	0.3	0.4	0.1
LT_LREAL	2.9	1.1	0.9	0.4	2.9	1.5	1.3	0.6	1.5	0.5	1.3	0.6	0.4	0.1
LE_INT	2.5	1.0	0.6	0.2	2.5	1.0	1.1	0.5	1.8	0.6	1.1	0.3	0.4	0.1
LE_DINT	2.3	1.0	0.6	0.2	2.4	1.0	1.0	0.4	1.6	0.5	1.0	0.3	0.4	0.1
LE_UINT	2.4	1.0	0.6	0.3	2.4	1.0	1.0	0.4	1.0	0.4	1.0	0.3	0.4	0.1
LE_REAL	2.5	1.0	0.6	0.2	2.4	1.0	1.1	0.4	1.6	0.7	1.0	0.3	0.4	0.1
LE_LREAL	2.9	1.1	0.9	0.3	2.9	1.5	1.3	0.6	1.4	0.5	1.3	0.6	0.4	0.1
Conversion														
to INT_BCD-4	2.1	1.0	0.5	0.2	2.1	0.7	1.0	0.4	1.5	0.7	1.0	0.3	0.3	0.1
to INT_DINT	1.9	1.0	0.4	0.2	1.4	0.5	0.9	0.4	1.3	0.5	0.9	0.3	0.3	0.1
to INT_UINT	2.9	1.0	0.4	0.2	1.7	0.7	0.8	0.4	1.2	0.5	0.9	0.4	0.3	0.1
to DINT_BCD-8	2.5	1.0	0.6	0.2	2.2	0.7	1.0	0.4	1.5	0.5	1.0	0.3	0.3	0.1
to DINT_INT	1.9	1.0	0.5	0.2	1.4	0.5	0.8	0.4	1.5	0.7	0.9	0.4	0.3	0.1
to DINT_UINT	1.9	1.0	0.4	0.2	1.6	0.6	0.8	0.4	1.3	0.5	1.0	0.3	0.3	0.1
to UINT_INT	1.9	1.0	0.4	0.2	1.4	0.5	0.8	0.4	1.3	0.5	1.0	0.3	0.3	0.1
to UINT_DINT	1.9	1.0	0.4	0.2	1.4	0.4	0.8	0.4	1.3	0.5	0.9	0.3	0.3	0.1
to UINT_BCD-4	2.1	1.0	0.5	0.3	2.0	0.7	0.9	0.4	1.4	0.5	1.0	0.3	0.3	0.1
to BCD-4_INT	2.2	1.0	0.5	0.2	1.8	0.5	0.9	0.4	1.4	0.5	1.0	0.3	0.3	0.1
to BCD-4_UINT	2.3	1.0	0.5	0.2	2.1	0.6	1.0	0.4	1.5	0.5	1.1	0.4	0.3	0.1
to BCD-8_DINT	2.3	1.0	0.5	0.2	1.8	0.4	1.0	0.5	1.4	0.5	1.1	0.3	0.3	0.1
REAL_TO_INT	2.7	1.0	0.6	0.2	2.0	0.5	1.1	0.4	1.5	0.5	1.1	0.4	0.3	0.1
REAL_TO_UINT	2.5	1.0	0.6	0.2	2.6	0.7	1.2	0.4	1.6	0.5	1.1	0.4	0.3	0.1
REAL_TO_LREAL	2.0	1.0	0.5	0.3	2.3	1.3	0.9	0.6	1.0	0.5	1.1	0.5	0.3	0.1
REAL_TO_DINT	2.7	1.0	0.6	0.2	2.1	0.5	1.1	0.4	1.6	0.5	1.2	0.4	0.3	0.1
INT_TO_REAL	1.9	1.0	0.4	0.2	1.4	0.5	0.8	0.4	1.3	0.5	0.9	0.3	0.3	0.1
UINT_TO_REAL	1.9	1.0	0.5	0.2	1.8	0.6	0.8	0.4	1.3	0.5	0.9	0.3	0.3	0.1
DINT_TO_REAL	1.9	1.0	0.4	0.2	1.4	0.5	0.8	0.4	1.3	0.5	0.9	0.3	0.3	0.1
DINT_TO_LREAL	2.0	1.0	0.5	0.3	2.0	1.4	0.9	0.6	1.0	0.5	0.9	0.5	0.2	0.1
REAL_TRUN_INT	2.0	1.0	0.5	0.2	2.1	1.0	0.9	0.4	-	-	0.9	0.3	0.3	0.1
REAL_TRUN_DINT	2.2	1.0	0.5	0.2	21.	1.0	0.9	0.4	-	-	0.9	0.3	0.3	0.1
DEG_TO_RAD_REAL	1.9	1.0	0.4	0.2	1.9	1.0	0.8	0.4	1.2	0.5	0.8	0.3	0.3	0.1
DEG_TO_RAD_LREAL	2.5	1.0	0.6	0.3	2.6	1.4	1.1	0.6	1.1	0.5	1.0	0.6	0.3	0.1
RAD_TO_DEG_REAL	1.9	1.0	0.4	0.2	1.9	0.9	0.8	0.4	1.2	0.5	0.8	0.3	0.3	0.1
RAD_TO_DEG_LREAL	2.4	1.0	0.6	0.3	2.5	1.4	1.1	0.6	1.1	0.5	1.0	0.6	0.3	0.1

	CPU310		CPU320		CPE010		CPE020		CRE020		CPE030 CRE030		CPE040 CRE040	
Function	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**
BCD4_TO_REAL	2.2	1.0	0.6	0.2	2.0	0.7	1.0	0.4	1.4	0.5	1.0	0.3	0.3	0.1
BCD8_TO_REAL	2.5	1.0	0.6	0.2	2.4	0.7	1.1	0.4	1.6	0.5	1.1	0.4	0.3	0.1
LREAL_TO_DINT	2.6	1.1	0.6	0.3	2.9	1.4	1.2	0.6	1.2	0.5	1.2	0.5	0.3	0.1
LREAL_TO_REAL	2.0	1.0	0.5	0.3	2.3	1.3	1.0	0.6	1.1	0.5	0.9	0.5	0.3	0.1
Data Move														
BLKCLR	1.9	1.0	0.5	0.2	2.3	1.0	0.9	0.4	1.2	0.5	0.8	0.2	0.2	0.1
BITSEQ	3.9	3.9	1.0	1.0	3.9	3.8	1.6	1.6	2.4	2.4	1.6	1.5	0.5	0.5
MOVE_BIT	2.8	1.2	0.8	0.2	3.1	1.3	1.3	0.5	1.7	0.6	1.2	0.3	0.4	0.1
MOVE_DINT	2.1	1.3	0.4	0.3	2.3	1.2	1.1	0.5	-	-	0.9	0.3	0.3	0.1
MOVE_INT	2.1	1.3	0.5	0.3	2.3	1.2	-	-	-	-	0.9	0.3	0.3	0.1
MOVE_WORD	2.2	1.3	0.5	0.2	2.4	1.2	1.0	0.6	1.6	0.6	0.9	0.3	0.3	0.1
MOVE_DWORD	2.2	1.3	0.5	0.3	2.3	1.2	1.1	0.5	1.6	0.6	0.9	0.3	0.3	0.1
MOVE_REAL	2.2	1.3	0.4	0.2	2.3	1.2	1.1	0.5	1.6	0.6	1.0	0.3	0.3	0.1
MOVE_LREAL	2.8	1.3	0.6	0.3	2.8	1.6	1.2	0.7	1.1	0.6	1.1	0.7	0.3	0.1
MOVE_UINT	-	-			2.3	1.2	1.0	0.5	-	-	-	-	-	-
BLKMOV_WORD	2.7	2.1	0.6	0.5	2.7	2.1	1.1	0.9	2.3	1.0	1.1	0.6	0.4	0.2
BLKMOV_DINT	2.9	2.1	0.7	0.5	3.0	2.1	1.2	0.9	-	-	1.2	0.7	0.4	0.2
BLKMOV_INT	2.7	2.1	0.6	0.5	2.7	2.1	1.2	0.9	-	-	1.1	0.5	0.4	0.2
BLKMOV_DWORD	2.9	2.1	0.7	0.5	2.9	2.1	1.2	0.9	2.4	1.0	1.2	0.6	0.4	0.2
BLKMOV_REAL	3.2	2.4	0.7	0.5	3.0	2.1	1.2	0.9	-	-	1.3	0.7	0.4	0.2
BLKMOV_UINT	2.7	2.1	0.6	0.5	2.7	2.1	1.1	0.9	2.3	1.0	1.1	0.6	0.4	0.2
DATA_INIT_ASCII	1.0	1.3	0.2	0.2	1.4	1.2	0.4	0.5	-	-	0.1	0.2	0.1	0.1
DATA_INIT_COMM	1.2	1.2	0.2	0.2	1.4	1.2	0.5	0.5	-	-	0.3	0.4	0.2	0.2
DATA_INIT_DLAN	1.3	1.3	0.3	0.2	1.4	1.2	0.6	0.5	-	-	0.2	0.2	0.1	0.1
DATA_INIT_DINT	1.0	1.2	0.2	0.2	1.2	1.2	0.5	0.5	-	-	0.1	0.2	0.1	0.1
DATA_INIT_DWORD	1.1	1.3	0.2	0.2	1.2	1.3	0.5	0.6	-	-	0.1	0.2	0.1	0.1
DATA_INIT_INT	1.0	1.3	0.2	0.2	1.2	1.2	0.4	0.5	-	-	0.1	0.2	0.1	0.1
DATA_INIT_REAL	1.0	1.2	0.2	0.2	1.2	1.2	0.5	0.5	-	-	0.1	0.2	0.1	0.1
DATA_INIT_LREAL	1.3	1.3	0.2	0.2	1.1	1.6	0.5	0.7	0.6	0.6	0.3	0.6	0.01	0.01
DATA_INIT_WORD	1.0	1.3	0.2	0.3	1.2	1.3	0.5	0.5	-	-	0.1	0.3	0.1	0.1
DATA_INIT_UINT	1.0	1.2	0.2	0.2	1.2	1.2	0.4	0.5	-	-	0.1	0.2	0.1	0.1
SWAP_WORD	2.6	1.3	0.5	0.2	2.6	1.2	1.1	0.5	1.6	0.6	1.1	0.3	0.3	0.1
SWAP_DWORD	2.6	1.3	0.8	0.2	2.7	1.2	1.2	0.5	1.7	0.6	1.1	0.3	0.3	0.1
SHFR_BIT	6.0	2.6	1.5	0.7	6.6	3.1	2.9	1.3	3.9	2.2	2.4	1.0	0.8	0.3
SHFR_WORD	7.0	4.7	2.1	1.5	7.5	5.1	3.2	2.1	3.7	2.6	3.3	2.4	1.1	0.8
SHFR_DWORD	7.0	4.7	2.0	1.5	7.8	5.2	3.3	2.2	3.8	2.6	3.3	2.5	1.2	0.8
Data Table														
SORT_INT	36.5	1.3	9.9	0.2	45.0	1.2	19.3	0.5	20.8	0.6	19.8	0.3	6.5	0.1
SORT_UINT	36.5	1.3	9.9	0.2	45.0	1.3	19.2	0.5	20.8	0.6	19.7	0.3	6.5	0.1
SORT_WORD	36.5	1.3	9.9	0.3	44.9	1.2	19.2	0.5	20.8	0.7	19.7	0.3	6.5	0.1
TBLRD_INT	3.7	1.5	0.9	0.3	4.0	1.8	2.0	0.9	2.5	0.9	1.4	0.4	0.6	0.3
TBLRD_DINT	3.7	1.6	0.9	0.4	4.0	1.8	2.1	0.9	2.5	0.9	1.5	0.4	0.6	0.3
TBLWRT_INT	3.8	1.5	9.4	0.4	4.4	1.5	1.8	0.7	2.7	0.8	1.6	0.4	0.6	0.2
TBLWRT_DINT	3.7	1.5	1.0	0.4	4.3	1.6	1.8	0.7	2.74	0.8	1.6	0.4	0.6	0.2
FIFORD_INT	3.9	1.7	0.9	0.4	4.1	1.7	1.8	0.7	2.7	0.8	1.6	0.6	0.6	0.2

	CPU310		CPU320		CPE010		CPE020		CRE020		CPE030 CRE030		CPE040 CRE040	
Function	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**
FIFORD_DINT	3.8	1.7	0.9	0.4	4.0	1.7	1.8	0.7	2.6	0.8	1.6	0.6	0.6	0.2
FIFOWRT_INT	3.2	1.2	0.8	0.3	3.2	1.2	1.3	0.5	2.2	0.6	1.2	0.3	0.4	0.1
FIFOWRT_DINT	3.2	1.2	0.8	0.3	3.2	1.2	1.3	0.5	2.2	0.6	1.2	0.2	0.4	0.1
LIFORD_INT	3.7	1.7	0.9	0.4	3.7	1.7	1.7	0.7	2.5	0.8	1.5	0.6	0.6	0.2
LIFORD_DINT	3.7	1.7	0.9	0.4	3.7	1.7	1.6	0.7	2.5	0.8	1.6	0.6	0.6	0.2
LIFOWRT_INT	3.2	1.2	0.8	0.3	3.2	1.2	1.3	0.5	2.2	0.6	1.2	0.2	0.4	0.1
LIFOWRT_DINT	3.2	1.2	0.8	0.3	3.3	1.2	1.3	0.5	2.2	0.6	1.2	0.2	0.4	0.1
LIFOWRT_DWORD	3.2	1.2	0.8	0.3	3.2	1.2	1.3	0.5	2.2	0.6	1.2	0.3	0.4	0.1
Array														
ARRAY_MOVE_BIT	4.1	2.0	1.0	0.5	3.9	1.7	1.8	0.8	2.8	1.0	1.6	0.5	0.5	0.2
ARRAY_MOVE_BYTE	3.2	1.9	0.8	0.5	3.2	1.6	1.4	0.8	2.6	1.0	1.3	0.5	0.5	0.2
ARRAY_MOVE_WORD	3.2	1.9	0.8	0.5	3.1	1.6	1.4	0.8	2.6	1.0	1.3	0.5	0.5	0.2
ARRAY_MOVE_DINT	3.2	1.9	0.8	0.5	3.1	1.6	1.4	0.8	-	-	1.3	0.5	0.5	0.2
ARRAY_MOVE_DWORD	3.2	1.8	0.8	0.5	3.1	1.6	1.4	0.8	2.5	0.9	1.3	0.4	0.5	0..2
ARRAY-MOVE_INT	3.2	1.9	0.8	0.5	3.1	1.6	1.4	0.8	-	-	1.3	0.5	0.5	0.2
ARRAY_MOVE_UINT	3.2	1.9	0.8	0.5	3.2	1.6	1.4	0.8	-	-	1.3	0.4	0.5	0.2
SRCH_BYTE	4.1	1.9	1.1	0.5	4.1	1.8	1.7	0.8	3.0	0.9	1.8	0.5	0.6	0.2
SRCH_WORD	4.1	1.8	1.0	0.5	5.1	1.8	2.1	0.8	2.7	0.9	1.8	0.5	0.6	0.2
SRCH_DWORD	4.4	1.9	1.0	0.5	4.0	1.8	1.8	0.8	2.8	0.9	1.7	0.5	0.6	0.2
ARRAY_RANGE_WORD	4.2	0.4	1.0	0.6	4.3	1.7	1.8	0.7	2.7	0.9	1.8	0.4	0.6	0.2
ARRAY_RANGE_DWORD	4.1	0.4	1.1	0.6	4.3	1.7	1.9	0.7	2.8	0.9	2.0	0.4	0.6	0.2
ARRAY_RANGE_DINT	4.2	0.4	1.0	0.6	4.3	1.7	1.9	0.7	-	-	1.8	0.4	0.7	0.2
ARRAY_RANGE_INT	4.3	0.4	1.0	0.6	4.7	1.7	2.0	0.8	-	-	1.9	0.4	0.6	0.2
ARRAY_RANGE_UINT	4.2	0.4	1.0	0.6	4.2	1.7	1.8	0.7	-	-	1.8	0.4	0.6	0.2
Math														
ADD_INT	2.1	1.2	0.6	0.3	2.1	1.1	0.9	0.5	1.5	0.6	1.1	0.4	0.3	0.1
ADD_DINT	2.1	1.1	0.5	0.3	2.2	1.1	0.9	0.5	1.6	0.6	1.2	0.6	0.4	0.2
ADD_REAL	2.1	1.2	0.5	0.3	2.1	1.1	0.9	0.5	1.5	0.6	1.0	0.4	0.3	0.1
ADD_LREAL	2.9	1.4	0.8	0.3	3.1	1.6	1.3	0.7	1.3	0.6	1.3	0.6	0.4	0.1
ADD_UINT	2.1	1.2	0.5	0.3	2.1	1.1	0.9	0.5	-	-	1.1	0.4	0.3	0.1
SUB_INT	2.1	1.2	0.5	0.3	2.1	1.1	0.9	0.5	1.6	0.6	1.0	0.4	0.3	0.1
SUB_DINT	2.2	1.2	0.5	0.3	2.2	1.1	0.9	0.5	1.6	0.6	1.0	0.4	0.3	0.1
SUB_REAL	2.1	1.2	0.5	0.3	2.3	1.1	0.9	0.5	1.5	0.6	1.0	0.4	0.3	0.1
SUB_LREAL	2.7	1.3	0.7	0.3	2.9	1.6	1.2	0.7	1.3	0.6	1.5	0.7	0.4	0.1
MUL_INT	2.1	1.2	0.5	0.3	2.1	1.1	0.9	0.5	1.6	0.6	1.0	0.4	0.3	0.1
MUL_DINT	2.1	1.2	0.5	0.3	2.2	1.1	0.9	0.5	1.5	0.6	1.1	0.4	0.3	0.1
MUL MIXED	2.1	1.1	0.5	0.3	2.1	1.1	0.9	0.5	1.6	0.6	1.0	0.4	0.3	0.1
MUL_REAL	2.1	1.2	0.5	0.3	2.3	1.1	0.9	0.5	1.5	0.6	1.0	0.4	0.3	0.1
MUL_LREAL	2.8	1.3	0.7	0.3	2.9	1.6	1.2	0.7	1.3	0.6	1.5	0.6	0.4	0.1
MUL_UINT	2.1	1.2	0.6	0.3	2.2	1.2	0.9	0.5	-	-	1.0	0.4	0.3	0.1
DIV_INT	2.2	1.1	0.5	0.3	3.0	1.1	1.1	0.5	1.6	0.6	1.0	0.4	0.3	0.1
DIV_DINT	2.3	1.2	0.5	0.3	3.1	1.1	1.0	0.5	1.6	0.6	1.0	0.4	0.3	0.1
DIV_REAL	2.3	1.1	0.5	0.3	2.2	1.1	1.0	0.5	1.6	0.6	1.0	0.4	0.3	0.1
DIV_LREAL	2.9	1.3	0.8	0.3	3.0	1.6	1.3	0.7	1.3	0.6	1.3	0.6	0.5	0.1
DIV_MIXED	2.4	1.1	0.5	0.3	2.5	1.1	1.2	0.5	-	-	1.1	0.4	0.3	0.1

	CPU310		CPU320		CPE010		CPE020		CRE020		CPE030 CRE030		CPE040 CRE040	
Function	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**
MOD_INT	2.2	1.2	0.5	0.3	2.3	1.2	1.0	0.5	1.6	0.6	1.0	0.4	0.3	0.1
MOD_DINT	2.2	1.2	0.5	0.3	2.3	1.1	1.0	0.5	1.6	0.6	1.0	0.4	0.3	0.1
MOD_UINT	2.3	1.1	0.5	0.3	2.4	1.1	1.0	0.5	-	-	1.0	0.4	0.3	0.1
ABS_INT	1.9	1.0	0.5	0.3	2.2	0.9	0.9	0.4	1.3	0.5	0.9	0.3	0.3	0.1
ABS_DINT	2.0	1.0	0.4	0.2	1.9	0.9	0.8	0.4	1.4	0.6	0.9	0.3	0.3	0.1
ABS_REAL	2.0	1.0	0.4	0.2	2.2	0.9	0.8	0.4	1.3	0.5	1.2	0.3	0.2	0.1
ABS_LREAL	2.3	1.0	0.6	0.3	2.5	1.4	1.1	0.6	1.1	0.5	1.0	0.5	0.3	0.1
SCALE_INT	3.4	1.8	0.9	0.4	3.0	1.7	1.3	0.8	-	-	1.5	0.6	0.5	0.2
SCALE_DINT	3.1	1.8	0.7	0.4	3.5	1.8	1.5	0.8	-	-	1.3	0.4	0.4	0.2
SCALE_UINT	2.9	1.8	0.7	0.4	3.2	1.7	1.4	0.7	-	-	1.2	0.4	0.4	0.2
SQRT_INT	2.4	1.0	0.6	0.2	2.5	0.9	1.1	0.4	1.5	0.5	1.1	0.3	0.3	0.1
SQRT_DINT	2.8	1.0	0.6	0.2	2.8	0.9	1.3	0.6	1.6	0.5	1.3	0.3	0.4	0.1
SQRT_REAL	1.9	1.0	0.4	0.2	2.1	1.0	0.8	0.4	1.3	0.5	0.9	0.3	0.2	0.1
SQRT_LREAL	2.3	1.0	0.6	0.3	2.5	1.3	1.1	0.6	1.1	0.5	1.2	0.5	0.3	0.1
Trigonometric														
SIN_REAL	2.2	1.0	0.5	0.2	2.4	0.9	1.0	0.4	1.4	0.5	1.0	0.3	0.3	0.1
SIN_LREAL	2.7	1.0	0.7	0.3	3.0	1.7	1.2	0.6	1.2	0.5	1.2	0.5	0.4	0.1
COS_REAL	2.2	1.0	0.5	0.2	2.5	0.9	1.0	0.4	1.4	0.5	1.0	0.3	0.3	0.1
COS_LREAL	2.7	1.0	0.7	0.3	2.8	1.3	1.2	0.6	1.2	0.5	1.2	0.5	0.4	0.1
TAN_REAL	2.3	1.0	0.6	0.2	2.3	0.9	1.0	0.4	1.4	0.5	1.1	0.3	0.3	0.1
TAN_LREAL	2.8	1.0	0.7	0.3	3.2	1.4	1.4	0.6	1.3	0.5	1.2	0.5	0.4	0.1
ASIN_REAL	2.9	1.0	0.6	0.2	2.9	0.9	1.3	0.4	1.6	0.5	1.2	0.3	0.4	0.1
ASIN_LREAL	3.1	1.1	0.9	0.3	3.2	1.4	1.4	0.6	1.4	0.5	1.6	0.5	0.4	0.1
ACOS_REAL	2.9	1.0	0.6	0.23	2.9	0.9	1.3	0.4	1.4	0.5	1.2	0.3	0.4	0.1
ACOS_LREAL	3.3	1.1	0.8	0.3	3.4	1.4	1.5	0.6	1.5	0.5	1.5	0.5	0.4	0.1
ATAN_REAL	2.2	1.0	0.5	0.2	2.4	0.9	1.0	0.4	1.7	0.5	1.0	0.3	0.3	0.1
ATAN_LREAL	2.7	1.1	0.7	0.3	2.9	1.4	1.2	0.6	1.3	0.5	1.2	0.5	0.4	0.1
Logarithmic														
LOG_REAL	2.4	0.9	0.6	0.2	0.2	0.9	1.0	0.4	1.4	0.5	1.0	0.3	0.3	0.1
LOG_LREAL	2.7	1.0	0.7	0.3	3.3	1.3	1.4	0.6	1.2	0.5	1.4	0.5	0.4	0.1
LN_REAL	2.5	0.9	0.6	0.2	0.2	0.9	1.0	0.4	1.4	0.5	1.0	0.3	0.3	0.1
LN_LREAL	2.7	1.0	0.7	0.3	3.3	1.4	1.4	0.6	1.3	0.5	1.4	0.5	0.4	0.1
EXPT_REAL	3.8	1.8	1.0	0.4	3.7	1.7	1.6	0.7	2.2	0.6	1.5	0.5	0.5	0.1
EXPT_LREAL	3.1	1.3	0.8	0.3	4.1	1.7	1.8	0.7	1.4	0.6	1.4	0.6	0.4	0.1
EXP_REAL	2.2	1.0	0.5	0.3	2.3	0.9	1.0	0.6	1.2	0.5	0.9	0.5	0.3	0.1
EXP_LREAL	2.7	1.0	0.7	0.3	3.4	1.4	1.4	0.6	1.3	0.5	1.4	0.6	0.4	0.1
PID														
PIDISA	6.8	6.1	1.6	1.4	7.4	6.6	3.1	2.8	4.1	3.7	2.5	2.3	0.9	0.9
PIDIND	6.8	6.1	1.7	1.4	7.2	6.5	3.1	2.8	4.0	3.7	2.5	2.3	0.9	0.8
Range														
RANGE_INT	3.3	1.8	0.8	0.4	3.3	1.8	1.4	0.8	2.2	1.0	1.5	0.7	0.5	0.2
RANGE_DINT	3.3	1.7	0.8	0.4	3.3	1.8	1.5	0.8	2.2	1.0	1.5	0.5	0.5	0.2
RANGE_DWORD	3.2	1.8	0.8	0.4	3.4	1.8	1.5	0.8	2.1	1.0	1.2	0.5	0.5	0.2
Timers														
ONDTR	4.7	3.6	1.1	0.9	4.7	3.7	2.0	1.6	2.7	2.2	1.8	1.3	0.6	0.4

	CPU310		CPU320		CPE010		CPE020		CRE020		CPE030 CRE030		CPE040 CRE040	
Function	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**
OFDT	4.5	4.0	1.0	0.9	4.6	4.0	2.0	1.7	2.7	2.4	1.9	1.6	0.6	0.5
TMR	4.5	4.0	1.1	0.9	4.7	4.1	2.0	1.8	2.6	2.4	1.7	1.4	0.6	0.5
TOF	7.8	4.7	1.8	1.2	9.6	4.9	4.1	2.1	NA	NA	3.9	2.0	1.3	0.6
TON	7.4	4.5	1.8	1.1	9.5	4.8	4.0	2.0	NA	NA	3.9	2.0	1.3	0.6
TP	7.5	4.5	1.8	1.2	9.8	4.8	4.2	2.1	NA	NA	3.9	2.0	1.3	0.6
Counters														
UPCTR	4.2	4.2	1.0	1.0	4.1	4.1	1.8	1.8	2.5	2.5	1.5	1.5	0.5	0.5
DNCTR	4.1	4.2	0.9	0.9	4.2	4.2	1.7	1.7	2.4	2.4	1.5	1.5	0.5	0.5
Control														
JUMPN	0.3	0.2	0.2	0.02	0.3	0.3	0.1	0.1	0.1	0.1	0.1	0	0	0
FOR/NEXT	1.5	0.8	0.3	0.1	1.4	0.6	0.7	0.3	0.6	0.4	0.1	0.1	0.1	0.1
MCRN/ENDMCRN Combined	0.8	0.9	0.1	0.1	0.7	0.7	0.3	0.3	0.3	0.3	0.4	0.3	0.1	0.1
SWITCH_POS	2.0	0.4	0.4	0.2	2.1	0.9	0.9	0.4	-	-	0.9	0.3	0.3	0.1
DOIO	56.8	1.2	37.7	2.4	38.0	1.2	19.8	0.6	51.2	0.6	15.4	0.3	9.4	0.1
DOIO with ALT	56.4	1.3	37.8	3.2	37.9	1.2	19.4	0.5	51.1	0.6	15.3	0.4	9.4	0.1
DRUM_SEQ	6.9	5.6	1.6	1.2	6.9	5.5	3.0	2.4	-	-	2.5	1.9	0.9	0.7
SCAN_SET_IO	153.6	1.7	105.6	0.4	56.6	1.9	33.3	0.8	33.1	0.9	30.0 30.3	0.8 0.9	22.7 22.8	0.3 0.3
SUSIO	2.3	0.3	0.5	0.1	2.6	0.4	1.2	0.2	1.1	0.3	0.9	0.1	0.4	0.1
COMMREQ	206.0	1.4	119.2	0.3	124.4	1.5	75.5	0.7	4.1	0.8	73.5	0.4	65.4	0.2
CALL/RETURN (LD)	7.8	0.3	1.7	0.1	9.1	0.6	4.0	0.3	5.0	0.3	2.8	0.1	1.0	0
CALL/RETURN (Parameterized Block)	4.9	0.4	1.2	0.1	6.3	0.6	2.7	0.3	3.2	0.2	2.0	0.1	0.7	0
CALL/RETURN (C Block)	7.3	0.4	1.8	0.1	12.3	0.6	5.0	0.3	5.1	0.3	3.7	0.1	1.3	0
Bus														
BUS_RD_BYTE*	NA	NA	NA	NA	25.7	2.5	12.5	1.0	10.9	0.6	8.9	0.8	5.5	0.3
BUS_RD_DWORD*	NA	NA	NA	NA	25.8	2.4	12.5	1.0	11.0	0.5	9.0	0.8	5.6	0.3
BUS_RD_WORD*	NA	NA	NA	NA	25.8	2.4	12.4	1.0	10.9	0.5	9.0	0.8	5.6	0.3
BUS_WRT_BYTE*	NA	NA	NA	NA	27.6	2.7	13.6	1.1	11.4	0.5	10.8	1.3	6.2	0.4
BUS_WRT_DWORD*	NA	NA	NA	NA	27.5	2.7	13.6	1.1	11.4	0.5	10.3	0.8	6.0	0.3
BUS_WRT_WORD*	NA	NA	NA	NA	27.7	2.7	13.7	1.0	11.4	0.5	10.3	0.7	6.0	0.3
BUS_RMW_BYTE*	NA	NA	NA	NA	27.1	2.9	14.0	1.1	12.1	0.4	10.8	0.8	6.8	0.3
BUS_RMW_DWORD*	NA	NA	NA	NA	26.5	2.9	14.0	1.1	12.3	0.4	10.4	0.9	6.8	0.3
BUS_RMW_WORD*	NA	NA	NA	NA	26.5	2.9	14.4	1.1	12.2	0.4	10.7	0.8	6.8	0.3
BUS_TS_BYTE*	NA	NA	NA	NA	25.3	2.3	13.8	0.9	12.0	0.1	10.1	0.5	6.7	0.2
BUS_TS_WORD*	NA	NA	NA	NA	25.9	2.3	13.8	0.9	12.2	0.1	10.2	0.5	6.7	0.2
	* Results will vary with how quickly the module responds to VME cycles. Because of this, incremental times do not appear in the table on page A-10. (These times were measured using a Series 90-70 Genius bus Controller.)													

	CPU310		CPU320		CPE010		CPE020		CRE020		CPE030 CRE030		CPE040 CRE040	
Function	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**	**Enabled**	**Disabled**
SVC_REQ														
#1 Change/Rd Constant Sweep Timer	8.5	1.1	1.2	0.1	32.0	1.1	13.9	0.5	31.9	0.5	16.3	0.3	5.3	0.1
#2 Rd Window Modes and Times	7.6	1.0	1.5	0.2	42.7	1.0	18.1	0.4	44.5	0.5	22.8	0.2	7.6	0
#3 Change Comms Window Mode and Timer Value	5.1	1.0	0.9	0.1	19.5	1.2	8.7	0.5	17.8	0.6	8.8	0.3	2.9	0.1
#4 Change Backplane Comms Window Mode and Timer Value	5.3	1.2	0.9	0.1	19.5	1.1	8.7	0.5	17.9	0.6	8.8	0.3	2.9	0.1
#5 Change Background Task Window Mode and Timer Value	5.1	0.9	0.8	0.1	19.5	1.1	8.7	0.5	18.0	0.6	8.8	0.3	2.9	0.1
#6 Change/Rd No of Words to Checksum	5.0	0.9	0.9	0.1	19.0	1.2	8.5	0.5	17.8	0.6	8.9	0.3	2.9	0.1
#7 Rd or Change the Time-of-Day Clock	8.7	1.0	1.8	0.2	10.3	1.2	4.5	0.5	4.9	0.7	3.9	0.4	1.3	0.1
#8 Reset Watchdog Timer	7.6	1.0	3.1	0.2	10.2	1.0	4.8	0.4	5.6	0.5	4.3	0.2	2.7	0.1
#9 Rd Sweep Time from Beg. of Sweep	4.9	1.0	1.1	0.2	7.2	1.0	3.3	0.4	3.5	0.5	2.8	0.2	1.0	0.1
#10 Rd Target Name	6.7	1.0	1.6	0.2	6.8	1.0	3.0	0.4	3.2	0.5	2.2	0.2	0.8	0.1
#11 Rd PLC ID	4.1	1.1	1.1	0.2	4.4	1.0	2.2	0.6	2.3	0.5	1.7	0.2	0.6	0
#12 Rd PLC Run State	2.5	1.1	0.6	0.2	2.8	1.0	1.1	0.4	1.6	0.5	0.9	0.2	0.3	0.1
#13 Stop PLC	4.2	0.9	0.8	0.1	18.1	1.1	7.9	0.5	17.0	0.6	8.5	0.2	2.8	0.1
#14 Clear PLC or I/O Fault Table	500.5	0.9	120.4	0.1	516.9	1.2	225.4	0.6	220.3	0.6	233.9	0.4	83.4	0.1
#15 Rd Last-Logged Fault Table Entry	13.8	1.1	0.6	0.3	3.7	1.1	1.5	0.5	9.0	0.5	3.8	0.3	0.4	0.1
#16 Rd Elapsed Time Clock	4.7	1.0	1.0	0.2	6.1	1.0	2.7	0.4	3.1	0.5	2.5	0.2	0.8	0.1
#17 Mask/Unmask I/O Interrupt	3.5	1.0	0.7	0.2	4.5	1.0	1.8	0.4	NA	NA	1.4	0.2	0.5	0.1
#18 Rd I/O Override Status	112.5	1.0	41.2	0.2	113.4	1.0	48.2	0.4	108.2	0.5	56.2	0.2	18.7	0.1
#19 Set Run Enable/Disable	4.4	1.0	1.0	0.2	17.6	1.0	7.6	0.4	17.1	0.5	8.1	0.2	2.8	0.1
#20 Rd Fault Tables	21.5	1.0	4.6	0.2	14.0	1.0	5.9	0.4	12.2	0.5	5.8	0.2	1.7	0.1
#21 User-Defined Fault Logging	34.5	1.1	8.5	0.2	51.8	1.0	24.0	0.4	33.5	0.5	20.4	0.3	8.9	0.1
#22 Mask/Unmask Timed Interrupts	2.8	1.0	0.6	0.2	3.2	1.0	1.3	0.4	1.9	0.5	1.0	0.3	0.3	0.1
#23 Rd Master Checksum	178.1	1.0	32.4	0.2	139.1	1.0	59.0	0.5	54.5	0.5	62.4	0.3	20.8	0.1
#24 Reset Module	58.1	1.1	1.4	0.2	187.1	1.0	88.7	0.4	-	-	38.5	0.3	35.5	0.1
#25 Disable/Enable EXE Block and Standalone C Program Checksums	2.9	1.1	0.8	0.2	5.0	1.0	2.2	0.6	2.4	0.5	1.9	0.3	0.7	0.1
#32 Suspend/Resume I/O Interrupt	-	-	-	-	59.9	1.0	51.0	0.4	NA	NA	47.7	0.3	45.2	0.1
#50 Rd Elapsed Time Clock	4.3	1.0	1.0	0.2	6.3	1.0	2.8	0.4	3.1	0.5	2.4	0.3	0.8	0.1
#51 Rd Sweep Time from Beg. of Sweep	4.6	1.0	1.1	0.2	7.1	1.0	3.2	0.5	3.5	0.5	2.8	0.3	1.0	0.1

Incremental Times

An Increment time is shown for functions that can have variable length inputs.

Incremental time is added to the base function time for each addition to the length of an input parameter. This time applies only to functions that can have varying input lengths (Search, Array Moves, etc.)

Units:

- For table functions, increment is in units of length specified.
- For bit operation functions, increment is microseconds per bit.
- For data move functions, microseconds per unit.

Function	*CPU310*	*CPU320*	*CPE010*	*CPE020*	*CRE020*	*CPE030/CRE030*	*CPE040/CRE040*
Bit Operation							
AND_WORD	0.12	0.03	0.11	0.05	0.05	0.05	0.02
AND_DWORD	0.16	0.03	0.17	0.07	0.07	0.07	0.02
OR_WORD	0.12	0.03	0.11	0.05	0.05	0.05	0.02
OR_DWORD	0.16	0.03	0.17	0.07	0.07	0.07	0.02
XOR_WORD	0.12	0.03	0.11	0.05	0.05	0.05	0.02
XOR_DWORD	0.16	0.03	0.17	0.07	0.07	0.07	0.02
NOT_WORD	0.08	0.02	0.09	0.04	0.04	0.04	0.01
NOT_DWORD	0.12	0.03	0.13	0.05	0.06	0.05	0.02
MCMP_WORD	0.26	0.06	0.22	0.10	0.10	0.09	0.03
MCMP_DWORD	0.29	0.06	0.25	0.11	0.11	0.1	0.03
SHL_WORD	0.17	0.05	0.17	0.07	0.07	0.07	0.02
SHL_DWORD	0.18	0.04	0.22	0.10	0.10	0.09	0.03
SHR_WORD	0.18	0.05	0.18	0.08	0.08	0.07	0.02
SHR_DWORD	0.19	0.05	0.23	0.10	0.10	0.1	0.03
BTST_WORD	0	0	0	0	0	0	0
BTST_DWORD	0	0	0	0	0	0	0
ROL_WORD	0.19	0.05	0.06	0.08	0.08	0.07	0.02
ROL_DWORD	0.17	0.02	0.20	0.09	0.09	0.08	0.03
ROR_WORD	0.16	0.04	0.20	0.08	0.08	0.08	0.03
ROR_DWORD	0.17	0.04	0.20	0.08	0.08	0.08	0.03
BPOS_WORD	0.76	0.2	0.7	0.3	0.30	0.28	0.09
BPOS_DWORD	1.69	0.4	1.54	0.66	0.66	0.61	0.20
Data Move							
MOVE_BIT	0.02	0.01	0.02	0.01	0	0.01	0
MOVE_INT	0.04	0.0	0.05	-	-	0.02	0.01
MOVE_DINT	0.04	0.01	0.05	0.02	0.02	0.02	0.01
MOVE_UINT	-	-	0.02	0.01	0.02	0.03	0.01
MOVE_WORD	0.02	0	0.02	0.01	0.01	0.01	0.0
MOVE_DWORD	0.04	0.01	0.05	0.02	0.02	0.01	0.01
MOVE_REAL	0.04	0.01	0.04	0.02	0.02	0.02	0.01
MOVE_LREAL	0.09	0.02	0.08	0.04	0.04	0.03	0.01
DATA_INIT_ASCII	0.01	0	0.01	0	0	0	0
DATA_INIT_COMM	0.04	0	0.02	0.01	0.01	0.01	0
DATA_INIT_DLAN	0	0	0	0	0	0.01	0

Function	CPU310	CPU320	CPE010	CPE020	CRE020	CPE030/CRE030	CPE040/CRE040
DATA_INIT_DINT	0.02	0.01	0.04	0.02	0.02	0.01	0
DATA_INIT_DWORD	0.04	0.01	0.04	0.02	0.02	0.01	0
DATA_INIT_INT	0.02	0	0.02	0.01	0.01	0.01	0
DATA_INIT_REAL	0	0.01	0.04	0.02	0.01	0.01	0
DATA_INIT_LREAL	0.08	0.02	0.09	0.03	0.03	0.03	0.01
DATA_INIT_WORD	0	0	0.02	0.01	0.01	0.01	0
DATA_INIT_UINT	0	0.0	0.02	0.01	0.01	0.01	0
SWAP_WORD	0.19	0.05	0.13	0.05	0.05	0.06	0.02
SWAP_DWORD	0.16	0.04	0.17	0.07	0.07	0.07	0.02
BLKCLR	0.03	0.06	0.06	0.01	0.02	0.01	0.0
SHFR_BIT	0.05	0.01	0.05	0.02	0.02	0.02	0.01
SHFR_WORD	0.16	0.04	0.19	0.08	0.09	0.08	0.03
SHFR_DWORD	0.20	0.05	0.23	0.09	0.10	0.09	0.03
Data Table							
SORT_INT	0.74	0.2	2.80	0.72	1.2	0.85	0.28
SORT_UINT	0.74	0.2	2.81	0.72	1.2	0.85	0.28
SORT_WORD	0.74	0.2	2.81	0.72	1.2	0.85	0.28
TBLRD_INT	0	0	0	0	0	0	0
TBLRD_DINT	0	0	0	0	0	0	0
TBLWRT_INT	0	0	0	0	0	0	0
TBLWRT_DINT	0	0	0	0	0	0	0
FIFORD_INT	0.02	0	0.02	0.01	0.01	0.01	0
FIFORD_DINT	0.04	0.01	0.05	0.02	0.02	0.02	0
FIFOWRT_INT	0	0	0	0	0	0	0
FIFOWRT_DINT	0.01	0	0	0	0	0	0
LIFORD_INT	0	0	0	0	0	0	0.01
LIFORD_DINT	0	0.9	0	0	0	0	0.01
LIFOWRT_INT	0	0	0	0	0	0	0
LIFOWRT_DINT	0	0	0	0	0	0	0
LIFOWRT_DWORD	0	0	0	0	0	0	0
Array							
ARRAY_MOVE_BIT	0.02	0	0.02	0.01	0.01	0.01	0
ARRAY_MOVE_BYTE	0.01	0	0.01	0	0	0.01	0
ARRAY_MOVE_INT	0.02	0	0.02	0.01	0.01	0.01	0
ARRAY_MOVE_DINT	0.05	0.01	0.05	0.02	0.02	0.02	0.01
ARRAY_MOVE_WORD	0.02	0.01	0.02	0.01	0.01	0.01	0
ARRAY_MOVE_DWORD	0.04	0.8	0.05	0.02	0.02	0.02	0.01
ARRAY_MOVE_UINT	0.02	0.8	0.02	0.01	0.01	0.01	0
SRCH_BYTE	0.07	0.02	0.05	0.02	0.02	0.02	0.01
SRCH_WORD	0.07	0.02	0.07	0.03	0.03	0.03	0.01
SRCH_DWORD	0.07	0.02	0.07	0.03	0.02	0.02	0.01
ARRAY_RANGE_DINT	0.54	0.1	0.58	0.25	0.23	0.26	0.09
ARRAY_RANGE_INT	0.52	0.1	0.55	0.24	0.22	0.25	0.08
ARRAY_RANGE_UINT	0.52	0.1	0.55	0.24	0.22	0.25	0.08
ARRAY_RANGE_WORD	0.52	0.1	0.56	0.24	0.26	0.26	0.09
ARRAY_RANGE_DWORD	0.56	0.1	0.61	0.26	0.27	0.27	0.09

Overhead Sweep Impact Times

This section contains overhead timing information for the PACSystems CPUs. This information can be used in conjunction with the estimated logic execution time to predict sweep times for the CPUs. The information in this section is made up of a base sweep time plus sweep impact times for each of the CPU models. The predicted sweep time is computed by adding the sweep impact time(s), the base sweep, and the estimated logic execution time.

A sample calculation for estimating sweep times is provided on page A-27.

The following components make up the total sweep time:

- Programmer communications sweep impact
- I/O Scan and fault sweep impact
- Ethernet Global Data sweep impact
- Intelligent Option Module (LAN modules) sweep impact
- I/O interrupt performance and sweep impact
- Timed interrupt performance and sweep impact

Base Sweep Times

Base sweep time is the time for an empty _MAIN program block to execute, with no configuration stored and none of the windows active. The following table gives the base sweep times in microseconds for each CPU model. For CPE and CRE models, base sweep time with I/O enabled includes time to scan the status bits for the Ethernet daughterboard.

Base Sweep Times

CPU Mode	RX3i		RX7i			
	CPU310	CPU320	CPE010	CPE020/ CRE020	CPE030 /CRE030	CPE040/ CRE040
Run I/O enabled	1086 µsec	180 µsec	457 µsec	182 µsec	169 µsec	77.4 µsec
Run I/O disabled	1076 µsec	176 µsec	449 µsec	180 µsec	165 µsec	74.8 µsec

The following diagram shows the differences between the full sweep phases and the base sweep phases.

Base Sweep vs. Full Sweep Phases

Base Sweep	***Full Sweep***
<START OF SWEEP>	<START OF SWEEP>
Sweep Housekeeping	Sweep Housekeeping
↓	↓
NULL Input Scan *	Input Scan *
↓	↓
Program Logic Execution	EGD Consumption Scan***
↓	↓
NULL Output Scan *	Program Logic Execution
↓	↓
↓	Output Scan *
↓	↓
↓	EGD Production Scans ***
↓	↓
↓	Poll for Missing I/O Modules **
↓	↓
↓	Controller Communications Window
↓	↓
↓	Backplane Communications Window
<END OF SWEEP>	<END OF SWEEP>

* If I/O is suspended, the input and output scans are skipped.
** Polling for missing I/O modules only occurs if a “Loss of ...” fault has been logged for an I/O module.
*** If no Ethernet Global Data (EGD) exchanges are configured, the consumption and production scans are skipped.

For the base sweep, if there is no configuration, the input and output scan phases of the sweep are NULL (i.e., check for configuration and then end). The presence of a configuration with no I/O modules or intelligent I/O modules (GBC) has the same effect. The logic execution time is not zero in the base sweep. The time to execute the empty _MAIN program is included so that you only need to add the estimated execution times of the functions actually programmed. The base sweep also assumes no missing I/O modules. The lack of programmer attachment means that the Controller Communications Window is never opened. The lack of intelligent option modules means that the Backplane Communications Window is never opened.

What the Sweep Impact Tables Contain

In some tables, functions are shown as asynchronously impacting the sweep. This means that there is not a set phase of the sweep in which the function takes place. For instance, the scanning of all I/O modules takes place during either the input or output scan phase of the CPU's sweep. However, I/O interrupts are totally asynchronous to the sweep and will interrupt any function currently in progress.

The communication functions (with the exception of the high priority programmer requests) are all processed within one of the two windows in the sweep (the Controller Communications Window and the Backplane Communications Window). Sweep impact times for the various service requests are all minimum sweep impact times for the defined functions, where the window times have been adjusted so that no time slicing (limiting) of the window occurs in a given sweep. This means that, as much as possible, each function is completed in one occurrence of the window (between consecutive logic scans). The sweep impact of these functions can be spread out over multiple sweeps (limited) by adjusting the window times to a value lower than the documented sweep impact time. For the programmer, the default time is 10 milliseconds; therefore, some of the functions listed in that section will naturally time slice over successive sweeps.

Programmer Sweep Impact Times

The following table shows nominal programmer sweep impact times in microseconds.

Programmer Sweep Impact Times

Sweep Impact Item	***Description***	***CPU310*** *(µsec)*	***CPU320*** *(µsec)*	***CPE010*** *(µsec)*	***CPE020 CRE020*** *(µsec)*	***CPE030 CRE030*** *(µsec)*	***CPE040 CRE0400*** *(µsec)*
Programmer window	The time required to open the Programmer Window but not process any requests. The programmer is attached through an Ethernet connection; no reference values are being monitored.	2.9	0.2	1.95	0.21	0.2	0.2
Reference table monitor	The sweep impact to refresh the reference table screen. (The %R table was used as the example.) Mixed table display impacts are slightly larger. The sweep impact may not be continuous, depending on the sweep time of the CPU and the speed of the host of the programming software.	4.9	0.29	1.2	0.33	0.26	0.29
Editor monitor	The sweep impact to refresh the editor screen when monitoring ladder logic. The times given in the table are for a logic screen containing one contact, two coils, and eleven registers. As with the reference table sweep impact, the impact may not be continuous.	4.1	0.31	1.41	0.35	0.31	0.31

I/O Scan and I/O Fault Sweep Impact

The I/O scan sweep impact has two parts, Local I/O and Genius I/O. The equation for computing I/O scan sweep impact is:

I/O Scan Sweep Impact = Local Scan Impact + Genius I/O Scan

Sweep Impact of Local I/O Modules

The I/O scan of I/O modules is impacted as much by location and reference address of a module as it is by the number of modules. The I/O scan has several basic parts.

I/O Scan	*Description*
Rack Setup Time	Each expansion rack is selected separately because of the addressing of expansion racks on the VME bus. This results in a fixed overhead per expansion rack, regardless of the number of modules in that rack.
Per Module Setup Time	Each Local I/O module has a fixed setup scan time.
Byte Transfer Time	The actual transfer of bytes is much faster for modules located in the main rack than for those in expansion racks. The byte transfer time differences will be accounted for by using different times for I/O modules in the main rack versus expansion racks.

In addition, analog input expander modules (the same as Genius blocks) have the ability to be grouped into a single transfer as long as consecutive reference addresses are used for modules that have consecutive slot addresses. Each sequence of consecutively addressed modules is called a scan segment. There is a time penalty for each additional scan segment.

RX7i I/O Module Types

Type	*Part Numbers*
Discrete Input Type I (16 point, 14 point)	IC697MDL240, IC697MDL241, IC697MDL251, IC697MDL640, IC697MDL671
Discrete Input Type II (32 point)	IC697MDL250, IC697MDL252, IC697MDL253, IC697MDL254, IC697MDL651, IC697MDL652, IC697MDL653, IC697MDL650, IC697MDL654
Discrete Output Type I (16 point, 12 point)	IC697MDL340, IC697MDL341, IC697MDL740, IC697MDL940
Discrete Output Type II (32 point)	IC697MDL350, IC697MDL750, IC697MDL752, IC697MDL753
Analog Input Type I (8 Channel)	IC697ALG230
Analog Input Type II (16 Channel with 8 channel AI module)	IC697ALG440, IC697ALG441
Analog Output (4 channel)	IC697ALG320

RX7i Module Sweep Impact Times (microseconds)

Note: The following table provides sweep impact times for modules in the Main rack and in an expansion (Exp) rack. The base case provides the overhead a single module in the rack. The increment (Inc) refers to the overhead for each similar module that is added to the same rack.

Model	***CPE010***				***CPE020/CRE020***				***CPE030/CRE030***				***CPE040/CRE040***			
Rack	***Main***		***Exp***		***Main***		***Expn***		***Main***		***Expn***		***Main***		***Exp***	
Impact	*Base*	*Inc*	*Base*	*Inc*	*Base*	*Inc*	*Base*	*Inc*	*Base*	*Inc*	*Base*	*Inc*	*Base*	*Inc*	*Base*	*Inc*
Discrete Input Type - I	35.8	9.3	29.6	30.2	9.5	6.3	10.2	13.8	10.2	10.6	14.4	16.5	5.1	4.1	6.0	9.8
Discrete Input Type - II	35.1	13.4	34.3	31.9	12.1	6.6	14.5	10.9	10.5	6.4	14.4	16.5	5.7	4.6	9.9	13.7
Discrete Output Type - I	38.9	14.8	33.1	33.0	12.9	6.2	11.7	14.4	11.5	6.4	10.8	12.9	6.0	4.5	6.3	10.0
Discrete Output Type - II	40.2	15.8	36.9	37.2	13.6	7.3	16.2	18.1	11.7	6.9	14.5	17.2	7.0	5.7	10.4	13.9
Per fault message	106.254	—	111.01	—	44.608	—	45.716	—								
Analog Input – Type 1	49.5	26.3	71.6	54.7	19.2	11.7	40.3	35.3	16.3	11.9	38	34	9.9	8.7	31.3	30.6
Analog Exp – Type 2	80.1	15.2	133.9	58.6	33.5	12.6	96.8	56.4	30.1	13.7	94.9	57	22.4	12.6	87.0	55.4
Analog Output	49.9	25.1	63.2	39.6	16.6	11.2	29.3	24.7	15.2	10.2	27	22.3	8.4	7.1	20.1	19.2
Per fault message	86.207	—	86.7	—	40.135		60.762	—								
Analog Input – IC697VAL132	74.0	54.3	N/A	N/A	44.9	38.2	N/A	N/A	42.8	38.1	N/A	N/A	35.8	32.2	N/A	N/A
Analog Input – IC697VAL264	91.4	74.8	N/A	N/A	61.7	59.6	N/A	N/A	60.9	57.8	N/A	N/A	53.6	48.2	N/A	N/A
Analog Input – IC697VRD008	86.5	38.0	N/A	N/A	33.4	33.4	N/A	N/A	38.8	28.4	N/A	N/A	41.2	37.0	N/A	N/A
Analog Output – IC697VAL301	96.5	67.0	N/A	N/A	52.3	47.5	N/A	N/A	45.1	42.3	N/A	N/A	43.0	38.7	N/A	N/A
Discrete Input – IC697VDD100	54.1	33.1	N/A	N/A	24.4	18.3	N/A	N/A	31.3	28.4	N/A	N/A	26.1	23.4	N/A	N/A
Discrete Output – IC697VDQ120	71.6	45.6	N/A	N/A	32.1	25.2	N/A	N/A	30.2	27.9	N/A	N/A	20.6	18.5	N/A	N/A
Discrete Output – IC697VDR151	87.2	70.6	N/A	N/A	36.1	34.0	N/A	N/A	32.2	29.2	N/A	N/A	28.7	23.5	N/A	N/A

RX3i I/O Module Types

Type	*Part Numbers*
Discrete Input (16 point)	IC694MDL240, IC694MDL241, IC694MDL645, IC694MDL646
Discrete Input (32 point)	IC694MDL654, IC694MDL655, IC694MDL654
Discrete Output (8 point)	IC694MDL330, IC694MDL732, IC694MDL930, IC694MDL940
Discrete Output (16 point, 12 point)	IC694MDL340, IC694MDL341, IC694MDL740, IC694MDL741
Discrete Output (32 point)	IC694MDL350, IC694MDL340, IC694MDL752, IC694MDL753 IC694MDL742, IC694MDL940
Discrete In/Out (8 point)	IC693MDR390, IC693MAR590
Analog Input (4 Channel)	IC695ALG220, IC694ALG221
Analog Input (16 Channel)	IC694ALG222, IC694ALG223
Analog Output (2 channel)	IC694ALG390, IC394ALG391

RX3i I/O Module Sweep Impact Times (microseconds)

Note: The following table provides sweep impact times for modules in the Main rack and in an expansion (Exp) rack. The base case provides the overhead for a single module in the rack. The increment (Inc) refers to the overhead for each similar module that is added to the same rack.

	CPU310				*CPU320*			
	Main Rack		*Expansion Rack*		*Main Rack*		*Expansion Rack*	
	Base	*Inc*	*Base*	*Inc*	*Base*	*Inc*	*Base*	*Inc*
Discrete Input 16 point	57.1	41.4	87.6	74.4	37.4	34.6	68.2	66.3
Discrete Input 32 point	78.4	59.7	105.9	96.1	56.2	55.3	86.1	85.7
Discrete Output 8 point	61.0	40.3	84.3	74.9	35.6	34.7	64.5	65.5
Discrete Output 16 point	61.5	38.9	87	74.4	35.4	34.5	65.2	64.9
Discrete Output 32 point	79.7	57	101.8	90.6	54.4	50.1	81.8	81.9
Discrete Mixed 8 point in/ 8 point out	104.5	85.7	167	151.7	72.2	68.9	132.3	131.2
Analog In/Out 4 channel	114.9	99	142.7	132	93.7	92.5	124.8	123.3
Analog Input 16 channel	427.7	407.1	538.8	538	385.3	378.8	499.9	499.3
Analog Output 2 channel	98.3	80.8	154.4	143.4	69.7	66.8	129.1	128.3
Universal Analog IC695ALG600	90.3	77.2	N/A	N/A	50.9	45.7	N/A	N/A
Analog Input 8 channel IC695ALG608	84.4	68.3	N/A	N/A	43.3	39.8	N/A	N/A
Analog Input 16 channel IC695ALG616	99.5	82.6	N/A	N/A	56.3	55.6	N/A	N/A
Analog Output 4 channel IC695ALG704	122	101.8	N/A	N/A	54.6	48.3	N/A	N/A
Analog Output 8 channel IC695ALG708	121.6	103.3	N/A	N/A	54.7	49.6	N/A	N/A

Worksheet A: I/O Module Sweep Time

The following form can be used for computing I/O module sweep impact. The calculation contains times for analog input expanders that are either grouped into the same scan segment as the preceding module or are grouped in a separate new scan segment. The sweep impact times can be found on page A-16.

Number of expansion racks			_____
Sweep impact per expansion rack	x	_____	= _____
Number of discrete I/O modules—main rack			_____
Sweep impact per discrete I/O module—main rack	x	_____	= _____
Number of discrete I/O modules—expansion rack			_____
Sweep impact per discrete I/O module—expansion rack	x	_____	= _____
Number of analog input base and output modules—main rack			_____
Sweep impact per analog input base and output module—main rack	x	_____	= _____
Number of analog input expander modules (same segment)—main rack			_____
Sweep impact per analog input expander module (same segment)—main rack	x	_____	= _____
Number of analog input expander modules (new segment)—main rack			_____
Sweep impact per analog input expander module (new segment)—main rack	x	_____	= _____
Number of analog input base and output modules—expansion rack			_____
Sweep impact per analog input base and output module—expansion rack	x	_____	= _____
Number of analog input base and output modules (same segment)—exp. rack			_____
Sweep impact per analog input base and output module (same seg.)—exp. rack	x	_____	= _____
Number of analog input base and output modules (new segment)—exp. rack			_____
Sweep impact per analog input base and output module (new seg.)—exp. rack	x	_____	= _____
Predicted I/O Module Sweep Impact			_____

Note: If point faults are enabled, substitute the corresponding times for point faults enabled, as shown in the following table.

An approximate per point or per channel average is shown in the following tables. These averages are based on 1024 points (512 in and 512 out) for discrete and 128 channels (96 in and 32 out) for analog. The 96 analog input channels consist of two base modules and five expanders. Actual values will vary from the approximate average, depending on the system I/O configuration.

Sweep Impact of Genius I/O and GBCs

For the sweep impact of Genius I/O and Genius Bus Controllers (GBC), there is a sweep impact for each GBC, a sweep impact for each scan segment, and a transfer time (per word) sweep impact for all I/O data.

The GBC sweep impact has three parts:

1. Sweep impact to open the System Communications Window. This is added only once when the first intelligent option module (of which the GBC is one) is placed in the system.
2. Sweep impact to poll each GBC for background messages (datagrams). This part is an impact for every GBC in the system.

Note: Both the first and second parts of the GBC's sweep impact may be eliminated by closing the Backplane Communications Window (setting its time to 0). This should only be done to reduce scan time during critical phases of a process to ensure minimal scan time. Incoming messages will timeout and COMM_REQs will stop working while the window is closed.

3. Sweep impact to scan the GBC. This results from the CPU notifying the GBC that its new output data has been transferred, commanding the GBC to ready its input data, and informing the GBC that the CPU has finished another sweep and is still in RUN mode.

A scan segment for a Genius I/O block consists of consecutive memory locations starting from a particular reference address. A new scan segment is created for each starting input or output reference address. The time to process a single scan segment is higher for an input scan segment than it is for an output scan segment. The scan segment processing is the same for analog, discrete, and global data scan segments. Discrete data is transferred a byte at a time and takes longer to complete the transfer than analog data, which is transferred a word at a time. Global data should be counted as either discrete or analog, based on the memory references used in the source or destination.

Sweep Impact Time of Genius I/O and GBCs

Note: Functions in **bold type** impact the sweep continuously. All other functions impact the sweep only when invoked. Not all the timing information listed in the following table was available at print time for this manual (the blank spaces).

	CPU310	***CPE010 (µsec)***	***CPE020 (µsec)***	***CPE030 (µsec)***	***CPE040 (µsec)***
Genius Bus Controller					
open backplane communications window	30	24	4	4	1
per Genius Bus Controller polling for background messages	403	19	11	9	6
per Genius Bus Controller I/O Scan					
Genius Bus Controller in the main rack	469	1	1	1	1
Genius Bus Controller in the expansion rack	683	11	7	6.9	1
Genius I/O Blocks					
per I/O block scan segment	3	217	217	193.7	208
per I/O block scan segment w/point faults enabled	3	217	217	194.8	213
per byte discrete I/O data in the main rack	13	3	3	2.1	3
per byte discrete I/O data in expansion racks	16	8	5	4.2	4
per word analog I/O data in the main rack	24	5	4	4	5
per word analog I/O data in expansion racks	34	11	8	8	11

Worksheet B: Genius I/O Sweep Time

Use the following worksheet for predicting the sweep impact due to Genius I/O. The sweep impact times can be found in “Sweep Impact Time of Genius I/O and GBCs,” above.

Open backplane communications window	_____	= _____
GBC poll for background messages Number of GBCs	_____ x _____	= _____
GBC I/O scan for the main rack	_____	
Number of GBCs in the main rack	X _____	= _____
GBC I/O scan for the expansion rack	_____	
Number of GBCs in the expansion rack	X _____	= _____
Input block scan segments—number of I/O block scan segments—sweep impact	_____ x _____	= _____
Output block scan segments—number of I/O block scan segments—sweep impact	_____ x _____	= _____
Bytes of discrete I/O data on GBCs—main rack Sweep impact/bytes of discrete I/O data—main rack	_____ x _____	= _____
Bytes of discrete I/O data on GBCs—expansion racks Sweep impact/bytes of discrete I/O data—expansion racks	_____ x _____	= _____
Words of analog I/O data on GBCs—main rack Sweep impact/word analog I/O data—main rack	_____ x _____	= _____
Words of analog I/O data on GBCs—expansion racks Sweep impact/word analog I/O data—expansion racks	_____ x _____	= _____
Predicted Genius I/O Scan Impact		_____

Ethernet Global Data Sweep Impact

Depending on the relationship between the CPU sweep time and an Ethernet Global Data (EGD) exchange's period, the exchange's data may be transferred every sweep or periodically after some number of sweeps. Therefore, the sweep impact will vary based on the number of exchanges that are scheduled to be transferred during the sweep. All of the exchanges must be taken into account when computing the worst-case sweep impact.

The Ethernet Global Data (EGD) sweep impact has two parts, Consumption Scan and Production Scan:

EGD Sweep Impact = Consumption Scan + Production Scan

This sweep impact should be taken into account when configuring the CPU constant sweep mode and setting the CPU watchdog timeout.

Where the Consumption and Production Scans consist of two parts, exchange overhead and byte transfer time:

Scan Time = Exchange Overhead + Byte Transfer Time

Exchange Overhead

Exchange overhead includes the setup time for each exchange that will be transferred during the sweep. When computing the sweep impact, include overhead time for each exchange.

Note: The exchange overhead times in the table below were measured for a test-case scenario of 1400 bytes over 100 variables.

EGD Exchange Overhead Time			
		Embedded Ethernet Interface	***Rack-based Ethernet Module***
CPU310	Consume / READ	NA	233.6 µsec
	Produce / WRITE	NA	480.6 µsec
CPU320	Consume / READ	NA	100.0 µsec
	Produce / WRITE	NA	195.1 µsec
CPE010	Consume / READ	184.3 µsec	238.2 µsec
	Produce / WRITE	342.0 µsec	452.0 µsec
CPE020	Consume / READ	87.7 µsec	117.8 µsec
	Produce / WRITE	187.9 µsec	257.5 µsec
CPE030	Consume / READ	85.1 µsec	114.1 µsec
	Produce / WRITE	191.8 µsec	253.5 µsec
CPE040	Consume / READ	35.08 µsec	47.12 µsec
	Produce / WRITE	75.16 µsec	103.0 µsec

Data Transfer Time

Note: This is the time required to transfer the data between the CPU module and the Ethernet module. EGD data transfer times do not increase linearly in relation to data size. Please use the data values in the table below to estimate data transfer times.

CPU	*Data Size (Bytes)*	*Direction*	*Embedded Ethernet Interface*	*Rack-based Ethernet Module*
CPU310	1	Consume / READ	NA	9.3 µsec
	100	Consume / READ	NA	51.8 µsec
	200	Consume / READ	NA	97.9 µsec
	256	Consume / READ	NA	123.8 µsec
	1	Produce / WRITE	NA	6.5 µsec
	100	Produce / WRITE	NA	14.1µsec
	200	Produce / WRITE	NA	17.7µsec
	256	Produce / WRITE	NA	19.3µsec
CPU320	1	Consume / READ	NA	6.2
	100	Consume / READ	NA	49.5
	200	Consume / READ	NA	96.4
	256	Consume / READ	NA	122.8
	1	Produce / WRITE	NA	3.4
	100	Produce / WRITE	NA	9.9
	200	Produce / WRITE	NA	14.9
	256	Produce / WRITE	NA	16.5
CPE010	1	Consume / READ	4.1 µsec	8.8 µsec
	100	Consume / READ	25.7µsec	23.5 µsec
	200	Consume / READ	49.0µsec	38.6 µsec
	256	Consume / READ	61.4µsec	46.8 µsec
	1	Produce / WRITE	1.9µsec	8.8 µsec
	100	Produce / WRITE	4.0µsec	16.5 µsec
	200	Produce / WRITE	6.0µsec	22.2 µsec
	256	Produce / WRITE	7.1µsec	25.1 µsec
CPE020	1	Consume / READ	2.7µsec	5.5 µsec
	100	Consume / READ	23.6µsec	19.5 µsec
	200	Consume / READ	46.3µsec	34.9 µsec
	256	Consume / READ	58.9µsec	42.7 µsec
	1	Produce / WRITE	0.8µsec	5.5 µsec
	100	Produce / WRITE	2.7µsec	13.9 µsec
	200	Produce / WRITE	4.7µsec	19.2 µsec
	256	Produce / WRITE	5.9µsec	22.1 µsec
CPE030	1	Consume / READ	2.8 µsec	5.3 µsec
	100	Consume / READ	25.8 µsec	18.7 µsec
	200	Consume / READ	50.7 µsec	33.4 µsec
	256	Consume / READ	60.1 µsec	40.4 µsec
	1	Produce / WRITE	0.8 µsec	5.5 µsec
	100	Produce / WRITE	2.5 µsec	13.1 µsec
	200	Produce / WRITE	4.2 µsec	18.2 µsec
	256	Produce / WRITE	5.2 µsec	21.5 µsec

CPU	Data Size (Bytes)	Direction	Embedded Ethernet Interface	Rack-based Ethernet Module
CPE040	1	Consume / READ	1.9µsec	3.85µsec
	100	Consume / READ	21.1µsec	10.1µsec
	200	Consume / READ	43.5µsec	31.4µsec
	256	Consume / READ	56.5µsec	39.2µsec
	1	Produce / WRITE	0.3µsec	3.8µsec
	100	Produce / WRITE	1.8µsec	11.8µsec
	200	Produce / WRITE	3.6µsec	16.8µsec
	256	Produce / WRITE	4.8µsec	19.8sec

Worksheet C: Ethernet Global Data Sweep Time

Number of consumed exchanges		_____		
Sweep impact per exchange	x	_____	=	_____
Number of data bytes in all of the consumed exchanges		_____		
Sweep impact per consumed data byte	x	_____	=	_____
Number of produced exchanges		_____		
Sweep impact per exchange	x	_____	=	_____
Number of data bytes in all of the produced exchanges		_____		
Sweep impact per produced data byte	x	_____	=	_____
		Predicted EGD Sweep Impact		_____

Sweep Impact of Intelligent Option Modules

The tables in this section list the fsweep impact times in microseconds for intelligent option modules. The fixed sweep impact is the sum of the polling sweep impact and the I/O scan impact. The opening of the Backplane Communications Window and the polling of each module have relatively small impacts compared to the sweep impact of CPU memory read or write requests.

Intelligent option modules include GBCs being used for Genius LAN capabilities. The sweep impact for these intelligent option modules is highly variable.

Fixed Sweep Impact Times of Intelligent Option Modules, RX7i

Sweep Impact Item	CPE010	CPE020	CPE030	CPE040
ETM (Peripheral Ethernet Module)	104 µsec	51 µsec	48 µsec	36 µsec
High Speed Counter	267 µsec	157 µsec	148 µsec	116 µsec
GBC	See “Sweep Impact Time of Genius I/O and GBCs,” page A-19.			

Fixed Sweep Impact Times of Intelligent Option Modules, RX3i

Sweep Impact Item	CPU310 (µsec)	CPU320
ETM (Peripheral Ethernet Module)	199	188 (one ETM module) 239 (two ETM modules)
High Speed Counter (APU300)	1085	1109
PROFIBUS Master		
No I/O	132	60
100 bytes Input, 100 bytes Output	196	105
100 bytes Input, 200 bytes Output	206	140
200 bytes Input, 100 bytes Output	248	106

I/O Interrupt Performance and Sweep Impact

There are several important performance numbers for I/O interrupt blocks. The sweep impact of an I/O interrupt invoking an empty block measures the overall time of fielding the interrupt, starting up the block, exiting the block, and restarting the interrupted task. The time to execute the logic contained in the interrupt block will affect the limit by causing the CPU to spend more time servicing I/O interrupts and thus reduce the maximum I/O interrupt rate.

The minimum, typical, and maximum interrupt response times reflect the time from when a single I/O module sees the input pulse until the first line of ladder logic is executed in the I/O interrupt block. Minimum response time reflects a 300 microsecond input card filter time + time from interrupt occurrence to first line of ladder logic in I/O interrupt block. The minimum response time can only be achieved when no intelligent option modules are present in the system and the programmer is not attached. Typical response time is the minimum response time plus a maximum interrupt latency of 2.0 milliseconds. This interrupt latency time is valid, except when one of the following operations occurs:

- The programmer is attached.
- A store of logic, RUN mode store, or word-for-word change occurs.
- A fault condition (logging of a fault) occurs.
- Another I/O interrupt occurs.
- The CPU is transferring a large amount of input (or output) data from an I/O controller (such as a GBC). Heavily loaded I/O controllers should be placed in the main rack whenever possible.

Any one of these events extends the interrupt latency (the time from when the interrupt card signals the interrupt to the CPU to when the CPU services the interrupt) beyond the typical value. However, the latency of an interrupt occurring during the processing of a preceding I/O interrupt is unbounded. I/O interrupts are processed sequentially so that the interrupt latency of a single I/O interrupt is affected by the duration of the execution time of all preceding interrupt blocks. (Worst case is that every I/O interrupt in the system occurs at the same time so that one of them has to wait for all others to complete before it starts.)

The maximum response time shown below does not include the two unbounded events.

I/O Interrupt Block Performance and Sweep Impact Times

Sweep Impact Item	*CPU310 (µsec)*	*CPU320 (µsec)*	*CPE010 (µsec)*	*CPE020 (µsec)*	*CPE030 (µsec)*	*CPE040 (µsec)*
I/O interrupt sweep impact	127.8	-	309.7	335	125.6	24
Minimum response time	151.7	326.1	392.4	334	330.6	315.2
Typical response time	175.0	327.3	396.1	336	331.5	315.5
Maximum response time	302.7	346.2	434.9	359	375.1	325.7

Note that the min, typical, and max response times include a 300 µsec Input card filter time.

Worksheet D: Programmer, IOM, I/O Interrupt Sweep Time

The following worksheet can be used to calculate the sweep impact times of programmer sweep impact, intelligent option modules, and I/O Interrupts. For time data, refer to the following tables:

Programmer Sweep Impact Times, page A-14

RX7i Module Sweep Impact Times, page A-16 or
RX3i I/O Module Sweep Impact Times (microseconds), page 17

Sweep Impact Time of Genius I/O and GBCs, page A-19

Programmer sweep impact			= _____
IOM—first module (open comm. window)		_____	
IOM—per module (polling)		+ _____	
LAN module I/O scan		+ _____	
	Total IOM Sweep Impact		= _____
CPU memory access from IOMs			= _____
I/O interrupt sweep impact		_____	
I/O interrupt response time		+ _____	= _____
	Predicted Sweep Time (Other)		_____

Timed Interrupt Performance

The sweep impact of a timed interrupt invoking an empty program block or timed program measures the overall time of fielding the interrupt, starting up the program or block, exiting the program or block, and restarting the interrupted task. The minimum, average, and maximum interrupt period reflect the time period from when the first line of ladder logic is executed in the timed interrupt block.

Timed Interrupt Performance and Sweep Impact Times for a 0.001s Timed Interrupt Block

Sweep Impact Item	***CPU310 (µsec)***	***CPU320 (µsec)***	***CPE010 (µsec)***	***CPE020 (µsec)***	***CPE030 (µsec)***	***CPE040 (µsec)***
Timed interrupt sweep impact	87.3	26.2	88.6	28	31.2	23.3
Minimum interrupt period	908.3	969.8	951.4	946	922.8	973
Average interrupt period	1000	1000.0	1005.5	999.7	1000.0	999.9
Maximum interrupt period	1081.2	1030.8	1056.6	1054	1077.0	1026.9

Example of Predicted Sweep Time Calculation

The sweep time estimate in this example does not include a time for logic execution. The calculated sweep is for normal sweep time with point faults disabled, and the programmer is not attached. The times used in the calculation are extracted from the following tables:

Base Sweep Times, page A-12

RX7i Module Sweep Impact Times, page A-16 or
RX3i I/O Module Sweep Impact Times (microseconds), page 17

Sweep Impact Time of Genius I/O and GBCs, page A-19

A sample calculation of predicted sweep times is provided after the example.

Sample RX7i System Configuration

PS	CPE010	BTM	32PT Input	32PT Input	32PT Output	32PT Output	8CHN Analog Input	4CHN Analog Output	ETM
0	1	2	3	4	5	6	7	8	9

MAIN RACK

Sweep Calculations

Predicted Sweep = Base Sweep + I/O Scan Impact

Base Sweep Time		= 465
I/O Scan Impact …		
Number of discrete input type 2 modules—main rack	2	
Sweep impact time per discrete I/O module	x 37.9	= 75.8
Number of discrete output type 2 modules—main rack	2	
Sweep impact time per discrete I/O module	x 80.4	= 80.4
Number of analog input modules—main rack	1	
Sweep impact time per analog base and output module	x 49.3	= 49.3
Number of analog output modules—main rack	1	
Sweep impact time per analog base and output module	x 49.7	= 49.7
Ethernet Module	1 x .55	= 55.0
Predicted Sweep Time		= 775.2

Note: Times are in microseconds.

Appendix B User Memory Allocation

User Memory Size is the number of bytes of memory available to the user for PLC applications.

Model	*User Memory Size*	*Bytes*
IC695CPU310, IC698CPE010, IC698CPE020, IC698CRE020	10MB	10,485,760
IC695CPU320	64MB	67,108,834
IC698CPE030, IC698CRE030 IC698CPE040, IC698CRE040	64MB	67,108,834

For a list of items that count against user memory, refer to page B-2.

Items that Count Against User Memory

The following items count against the CPU memory and can be used to estimate the minimum amount of memory required for an application. Additional space may be required for items such as Advanced User Parameters, zipped source files, user heap, and published symbols.

Register Memory Size (%R)	Bytes = %R references configured × 2
Word Memory Size (%W)	Bytes = %W references configured × 2
Analog Inputs (%AI)	If point faults enabled: Bytes = %AI references configured × 3 If point faults disabled: Bytes = %AI references configured × 2
Analog Outputs (%AQ)	If point faults enabled: Bytes = %AQ references configured × 3 If point faults disabled: Bytes = %AQ references configured × 2
Discrete Point Faults	If point faults enabled: Bytes = 3072
Managed Memory (Symbolic Variable and I/O Variable Storage)	The total number of bytes required for symbolic and I/O variables. Calculated as follows: [(number of symbolic discrete bits) * 3 / (8 bits/byte)] + [(number of I/O discrete bits) * Md / (8 bits/byte)] + [(number of symbolic words) * (2 bytes/word)] + [(number of I/O words) * (Mw bytes/word)] Md = 3 or 4. The number of bits is multiplied by 3 to keep track of the force, transition, and value of each bit. If point faults are enabled, the number of I/O discrete bits is multiplied by 4. Mw = 2 or 3. There are two 8-bit bytes per 16-bit word. If point faults are enabled, the number of bytes is multiplied by 3 because each I/O word requires an extra byte.
EGD (included in HWC)	Bytes = 0 if no Ethernet Global Data pages are configured
I/O Scan Set File (included in HWC)	Based on number of scan sets used. **Note:** 32 bytes of user memory are consumed if the application scans all I/O every sweep (the default).
User Programs	See “User Program Memory Usage” page B-3 for details on user programs.

User Program Memory Usage

Space required for user logic includes the following items.

%L and %P Program Memory

%L and %P are charged against your user space and sized depending on their use in your applications. The maximum size of %L or %P is 8192 words per block.

The %L and %P tables are sized to allow extra space for Run Mode Stores according to the following rules.

- If %L memory is not used in the block, the %L memory size is 0 bytes. If %L memory is used in the block, a buffer is added beyond the highest %L address actually used in logic or in the variable table. The default buffer size is 256 bytes, but can be changed by editing the Extra Local Words parameter in the block Properties.
- The same rules apply for the size of %P memory, but %P memory can be used in any block in the program.
- The buffer cannot make the %P or %L table exceed the maximum size of 8,192 words. In such a case, a smaller buffer is used.
- You can add, change, or delete %L and/or %P variables in your application and Run Mode Store the application if these variables fit in the size of the last-stored %L/%P tables (where the "size" includes the previous buffer space), or if going from a zero to non-zero size.
- The size of the %L/%P tables is always recalculated for Stop Mode Stores.

Program Logic and Overhead

The data area for C (.gefelf) blocks is considered part of the user program and counts against the user program size. Additional space is required for information internal to the CPU that is used for execution of the C block.

The program block is based on overhead for the block itself plus the logic and register data being used (that is, %L).

Note: The LD program’s stack is not counted against the CPU’s memory size.

Note: If your application needs more space for LD logic, consider changing some %P or %L references to %R, %W, %AI, or %AQ. Such changes require a recompilation of the program block and a Stop Mode store to the CPU.

Appendix C

Converting Series 90 Applications to PACSystems

PACSystems controllers incorporate the functional features of the Series 90 PLC family with the Ethernet Global Data (EGD) capabilities of the Series 90-30 PLC family and improved Ethernet communications. PACSystems provides many enhancements compared to the Series 90 PLCs, although some Series 90 functionality is not supported in the current version.

This chapter provides the following information:

PACSystems - Series 90 Comparison C-2

Converting an Application from Series 90-70 to PACSystems C-22

Converting an Application from Series 90-30 to PACSystems C-28

For CPU specifications, refer to chapter 2. For function timing information, refer to Appendix A.

PACSystems – Series 90 Comparison

This section summarizes differences in features and operation between the two control systems.

CPU Operation

Category	*Series 90-70*	*Series 90-30*	*PACSystems*
CPUs allowed in any rack 0 slot.	Not supported.	Not supported.	Supported by RX3i (requires two slots). Not supported by RX7i.
Logic Execution	Column major (See "Logic Execution," below.)	Row major	Row major (See "Logic Execution," below.)
Fault clearing	Clear individual faults.	Clear individual faults.	Clear the controller fault table or I/O fault table.
Reset of IO Module fault	Generated on store or clear of HWC.	Not generated on store or clear of HWC.	Generated on store or clear of HWC.
%S0020 status bit	Not supported.	Set ON when a relational function using REAL data executes successfully. Cleared when either input is NaN (Not a Number).	Not supported. In LD logic, the OK output can be used to indicate the function has executed successfully.
Access Privileges	Levels 0 through 4.	Levels 1 through 4	Levels 1 through 4
Stack overflow	Transitions to Stop/Halt mode when it detects a stack overflow.	Transitions to Stop/Fault mode when it detects a stack overflow.	If there is not enough stack space left to support a given block call, an "Application Stack Overflow" fault is logged. In these circumstances, the CPU cannot execute the block. Instead, it sets the block's Boolean outputs to FALSE, and resumes execution at the point after the block call instruction.
Program name	Not used	Not used	Read-only LD program name, LDPROG1, used by legacy drivers in Plant Edition software.
Low battery indication	Supported via the %SA0011 system status bit and the PLC fault table.	Supported via the %SA0011 system status bit and the PLC fault table.	Not supported. For details, see "Battery Status (Group 18)" in Chapter 15.

Logic

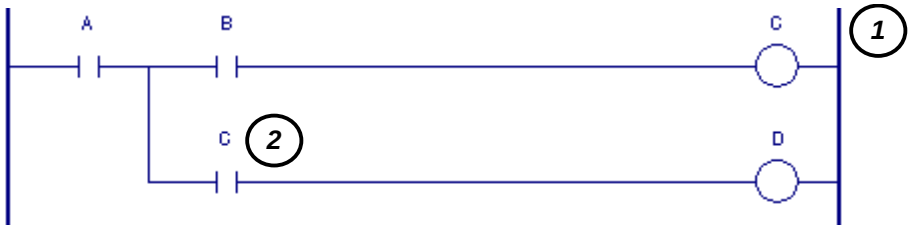
Row- vs. Column-major LD Execution

Series 90-70 PLCs use column major execution: they execute a rung of LD logic by going from top to bottom, left to right, through each column in the rung. Series 90-30, VersaMax, and PACSystems use row major execution: they execute an LD rung by tracing paths from left to right and top to bottom. Differences in execution order result from branching, or the divergence and/or convergence of power flow within a rung, which is not allowed in Series 90-30. Therefore, the following examples do not apply to Series 90-30.

Note: The conversion of a Series 90-70 target to a PACSystems target does not rewrite the logic from column major execution to row major execution. Rungs that may execute differently because of the column/row major difference are reported in the target conversion report, but it is not guaranteed that every execution difference will be detected and reported.

Example 1

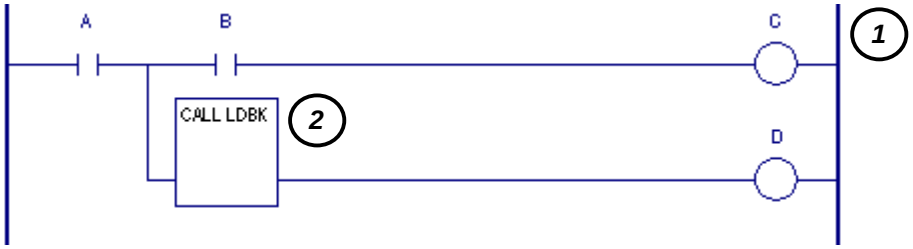
In this example, the order in which the contact and coil that reference the variable C are executed differs between column-major and row-major execution. In row-major, the coil (1) sets the value of C before the contact (2) is evaluated. In column-major, the C contact (2) is evaluated before the coil (1) is executed.



Example 2

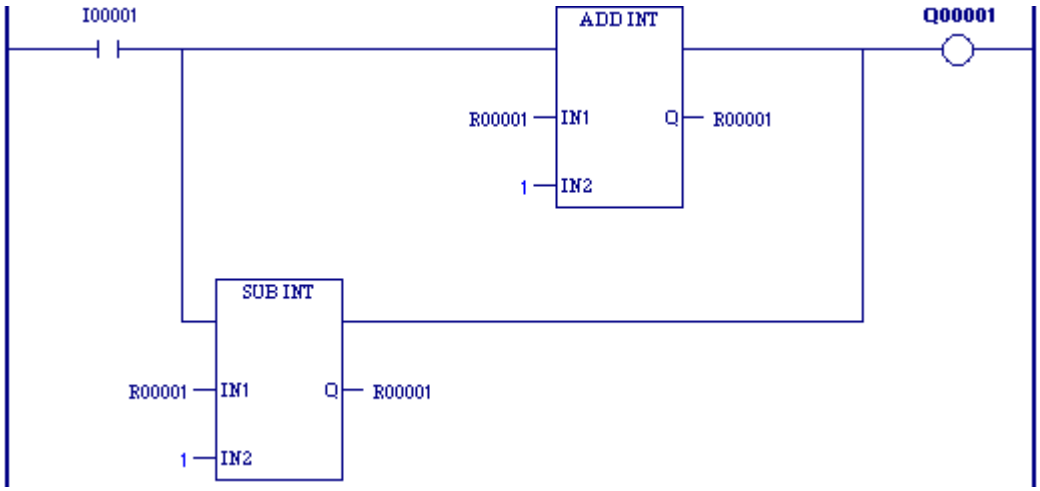
In this example, the problem is less obvious. If the variable C is used as an input and/or output inside LDBK or any block that LDBK calls, the difference in row-major versus column-major execution within the called blocks may affect the execution of coil C.

In row-major execution, the C coil (1) is executed before the Call (2) is executed. In column-major execution, the Call (2) is executed before the C coil (1) is executed.



Example 3

In the following example, using column-major execution (Series 90-70), the SUB_INT always executes before the ADD_INT. Using row-major execution (PACSystems), the ADD_INT executes before the SUB_INT.

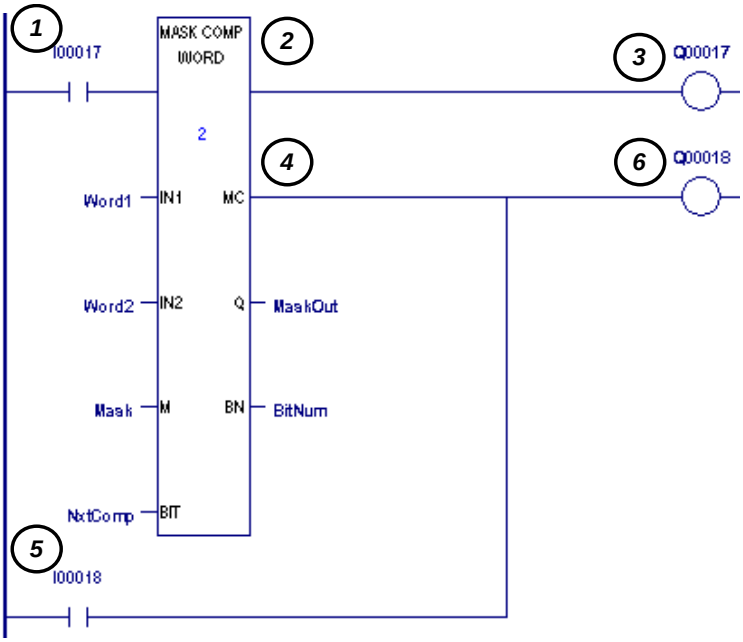


Example 4

In this example, even though the order of execution is different, the outcome is the same for both types of execution.

Row Major Execution (PACSystems)

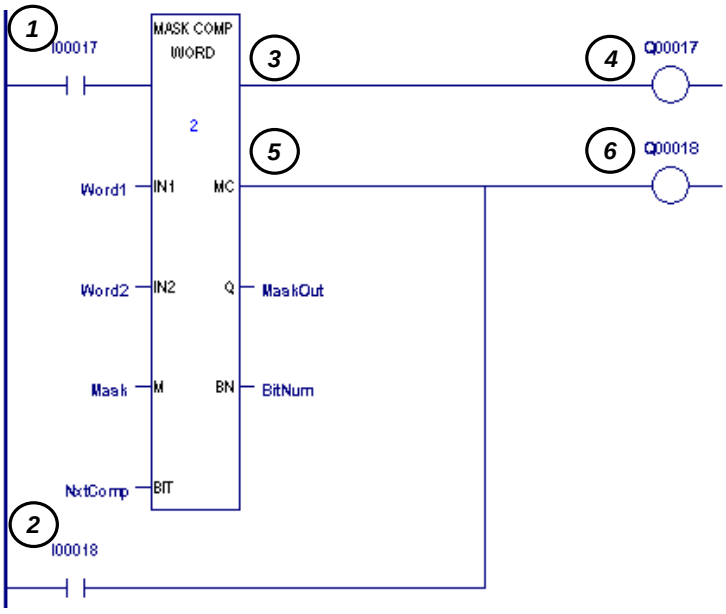
The external input I00017 (1) is evaluated first. If I00017 is set to 1, the Masked Compare function (2) is executed, passing power to Q00017 (3). If there is a miscompare, the output MC (4) is set to 1. I00018 (5) is then evaluated. The state of Q00018 (6) is set to the OR of the MC output and I00018.



Column Major Execution (Series 90-70)

The external input I00017 (1) is evaluated first, followed by external input I00018 (2). If I00017, is set to 1, the Masked Compare function (3) is executed, passing power to Q00017 (4). If there is a miscompare, the output MC (5) is set to 1.

The state of Q00018 (6) is set to the OR of the MC output and I00018.



Maximum Number of Blocks

Category	Series 90-70	Series 90-30	PACSystems
Maximum number of blocks	256, including _MAIN	65, including _MAIN	512, including _MAIN

User Programs

Category	Series 90-70	Series 90-30	PACSystems
C Standalone Programs	Supported	Not supported	Not supported
LD Program	Supported	Supported	Supported
Structured Text	Not supported.	Not supported.	For availability on a given CPU version, refer to the *IPI* document provided with that CPU.
Program Scheduling	Five modes are supported.	Only the Ordered mode is supported.	Only the Ordered mode is supported.
Interrupt programs	Supported	Not supported	Not supported
Function Blocks	Supported	Not supported.	For availability on a given CPU version, refer to the *IPI* document provided with that CPU.
Sequential Function Chart programming	Supported	Supported	Not supported
Synchronous Scan Sets	Supported	Not supported	Not supported
FIP, Microcycle mode and periodic programs	Supported	Not supported	Not supported
Multiple programs per folder	Supported (up to 16)	Not supported	Not supported
C Debugger	Supported	Not supported	Not supported
State Logic	Supported	Supported	Not supported

Stack Size

Category	Series 90-70	Series 90-30	PACSystems
Stack Size	Valid range: 1 through 64 KB Default: 20 KB (LD Program)	Not configurable.	Valid range: 8 through 320 KB, in increments of 8 KB Default: 64 KB

C Blocks

Category	Series 90-70	Series 90-30	PACSystems
C toolkit used to develop	GFK-0646	GFK-0646	GFK-2259
Bits	16-bit blocks	16-bit blocks	32-bit blocks
Compiled block extension	.exe or .sta	.exe	.gefelf
Maximum size	64,000 bytes	81,920 bytes	Limited only by available user memory. **Note:** By default for any C block, if its size is greater than 256 KB, a validation error takes place. This may be prevented by right-clicking the C block, choosing Properties and, in the Inspector, setting the Check Size Limits property to False. This, however, introduces the risk of memory fragmentation.
Importing	Cannot import blocks developed for PACSystems or Series 90-30	Cannot import blocks developed for PACSystems or Series 90-70	Cannot import blocks developed for Series 90-70 or Series 90-30. They must be recompiled with a .gefelf extension.
Name property	Up to 7 characters long	Up to 7 characters long	Up to 31 characters long
Check Size Limits	Not supported.	Not supported.	Supported.
Third-party VME interrupts	Supported.	Not supported.	Replaced with Module interrupts
Module interrupts	Not supported	Not supported.	Selected from the list of Triggers when you schedule the block. First, configure the module in the Hardware Configuration; then set a module's interrupt ID in the Interrupts tab of the Parameter editor.
Number of defined parameters	0 through 7 input/output pairs. Must have as many inputs as outputs.	Not supported.	Input: 0 through 63. Output: 1 through 64. Does not need to have as many inputs as outputs.
Optional parameters optional in the CALL to the C block	Not supported. You must supply a value for every defined parameter.	Not supported.	Machine Edition enables you to leave the value of any parameter blank. It is your responsibility to ensure the C block uses an acceptable default value and no run time error occurs.
BYTE data type	Supported	Not supported.	Refer to the *Important Product Information* document for a specific CPU firmware version.
NWORD data type	Supported.	Not supported.	Not required. Can use WORD instead.
Data flow	Not supported.	Not supported.	Supported
Indirect references	Not supported.	Not supported.	Supported
Bit references in non-discrete memory	Not supported	Not supported.	Supported. Must be byte-aligned.

Blocks

A block can be of type Block, Parameterized Block, or Function Block.

Blocks

Category	Series 90-70	Series 90-30	PACSystems
Maximum size	32 KB	16 KB	128 KB
Name property	Up to 7 characters long	Up to 7 characters long	Up to 31 characters long
Extra Local Words allocated for %P or %L memory per program block	Not configurable	Not supported	Configurable.
Third-party VME interrupts	Supported	Not supported	Replaced with Module interrupts
Module interrupts	Not supported	Not supported	Selected from a list when scheduling the block. First you need to configure the module and the interrupt in the Hardware Configuration.
Bit references in non-discrete memory	Not supported	Not supported	Supported

Parameterized Blocks

Referred to as *parameterized subroutine blocks* (PSBs) in Series 90-70. Not supported in Series 90-30.

Category	Series 90-70	PACSystems
BYTE data type	Supported	Refer to the *Important Product Information* document for a specific CPU firmware version.
NWORD data type	Supported	Not required. Can use WORD instead.
Y0 parameter	Cannot be used in the logic of a zero-parameter parameterized block. Y0 is assumed to be set to True.	Can be used in the logic of any parameterized block, including one with no parameters. Can also be used in a program block, except for _MAIN.
BOOL parameters	Flow not supported into a BOOL input parameter whose length is greater than 1, or out of a BOOL output parameter whose length is greater than 1. Only power flow supported; not data flow.	Data and power flow supported.
	Constants are not supported.	Constants are supported for input parameters.
32-bit parameters (DINT, DWORD, and REAL)	Constants are not supported.	Constants are supported for input parameters.
Bit references in non-discrete memory	Not supported	Supported
Block name	Maximum of 7 characters	Up to 31 characters long.
Parameters	Maximum of 7 inputs and 7 outputs. (8 including Y0 output)	Maximum of 63 inputs and 63 outputs (64 including Y0 output). Refer to the *Important Product Information* document for a specific CPU firmware version.

Function Blocks

PACSystems allows the use of Function Blocks, which are user-defined logic blocks that have parameters and instance data. Series 90 PLCs do not support this feature.

PACSystems Functions

The following functions were introduced with PACSystems and may not be supported in all Series 90 and VersaMax PLC product families.

BUS instructions

- BUS_RD_BYTE. Replaces the Series 90-70 VME_RD_BYTE function.
- BUS_RD_DWORD
- BUS_RD_WORD. Replaces the Series 90-70 VME_RD_WORD function.

Note: The BUS_RD_ instructions replace the Series 90-70 VME_CFG_READ function.

- BUS_RMW_BYTE. Replaces the Series 90-70 VME_RMW_BYTE function.
- BUS_RMW_DWORD
- BUS_RMW_WORD. Replaces the Series 90-70 VME_RMW_WORD function.
- BUS_TS_BYTE. Replaces the Series 90-70 VME_TS_BYTE function.
- BUS_TS_WORD. Replaces the Series 90-70 VME_TS_WORD function.
- BUS_WRT_BYTE. Replaces the Series 90-70 VME_WRT_BYTE function.
- BUS_WRT_DWORD
- BUS_WRT_WORD. Replaces the Series 90-70 VME_WRT_WORD function.

Note: The BUS_WRT_ instructions replace the Series 90-70 VME_CFG_WRITE function.

New Transitional Coils and Contacts

- NTCOIL
- NTCON
- PTCOIL
- PTCON

The status of the new transitional contacts PTCON and NTCON is determined by the value that the associated BOOL variable had the last time the contact was executed. The status of the existing POSCON and NEGCON transitional contacts is determined by the last write to the BOOL variable associated with the contact.

Service Requests

The PACSystems SVC_REQ function supports the following services:

- #50: Read Elapsed Time Clock (Two DWORDs)
- #51: Read Sweep Time from Beginning of Sweep (DWORD)

Function Differences

Most function differences between PACSystems and other PLC families, such as Series 90 and VersaMax, are due to the PACSystems CPUs' support of the following new features:

- Symbolic variables
- Bit addressing in non-discrete memory
- BOOL arrays of sufficient length can replace operands of other data types.

The following table lists other differences between Series 90-70 and PACSystems RX7i with regards to individual built-in functions.

Category	***Series 90-70***	***Series 90-30***	***PACSystems***
ARRAY_MOVE_ (all mnemonics)	The SR parameter does not support data flow.	Same as Series 90-70.	The SR parameter supports data flow.
ARRAY_RANGE_ (all mnemonics)	The LL, UL, IN, and Q parameters do not support data flow.	Function not supported.	The LL, UL, IN, and Q parameters support data flow. When an invalid (reference out of range) operand occurs and length is equal to one the result is set to false
BIT_SEQ	The N (STEP) parameter does not support data flow.	Same as Series 90-70.	The N parameter supports data flow.
Bit Operation Functions	Overlapping input and output reference address ranges in multi-word functions may produce unexpected results.	Same as Series 90-70.	When using overlapping inputs and outputs, the PACSystems performs the operation on the data when the function is invoked, not the data output from earlier executions.
COMM_REQ	Supports WAIT mode COMM_REQs.	Same as 90-70.	Does not support WAIT mode COMM_REQs.
	A smart module can update the status word if you have level 3 or level 4 security access.	Same as 90-70.	Level 2 is sufficient, except when a Series 90-70 GBC is involved. In this case, level 3 or 4 is required.
	Can use COMM_REQs with serial and Ethernet ports.	Same as 90-70.	Using COMM_REQs for communication with serial and Ethernet ports supported. Refer to the *Important Product Information* document for a specific CPU firmware version.
	System IDs address modules located in double-width slots.	Same as 90-70.	System IDs address modules located in single-width slots (RX7i).
DATA_INIT_ASCII DATA_INIT_COMM DATA_INIT_DINT DATA_INIT_DWORD DATA_INIT_INT DATA_INIT_REAL DATA_INIT_UINT DATA_INIT_WORD DATA_INIT_DLAN	The Q output parameter does not support indirect references or data flow.	Function not supported.	The Q output parameter supports indirect references and data flow.

Category	*Series 90-70*	*Series 90-30*	*PACSystems*
END	Not supported	Supported	Not supported
Enhanced DO_IO	Not supported	Supported	Not supported
RANGE_ (all mnemonics)	The L1, L2, and IN parameters do not support data flow.	Same as Series 90-70.	The L1, L2, and IN parameters support data flow.
SVC_REQ 5 Change Background Task Window Mode and Timer Value	Supported.	Not supported	Supported
SVC_REQ 6 Change/Read Number of Words to Checksum	Number of words to be checksummed is rounded to a multiple of 8. (Same as PACSystems.)	Number of words to be checksummed must be in the range 0 to 32.	Number of words to be checksummed is rounded to a multiple of 8.
SVC_REQ 13 Shut down (stop CPU)	Parameter block is ignored. (You must specify a dummy parameter, which SVC_REQ 13 does not use.) Stops at the end of the next scan. To mimic this behavior in PACSystems version 2.00 or later, use a value of –1 in the SVC_REQ parameter block and set Number of Last Scans in HWC to 0.	Parameter block is ignored. Always runs one scan before stopping CPU. To mimic this behavior in PACSystems version 2.00 or later, use a value of –1 in the SVC_REQ parameter block and set Number of Last Scans in HWC to 1.	For CPUs with firmware version 2.00 or later, 0 to 5 scans are allowed. A value of –1 causes the Number of Last Scans specified in HWC to be used. For CPUs with firmware versions earlier than 2.00, the value in the parameter block must be 0.
SVC_REQ 15 Read Last-Logged Fault Table Entry	Supports Read Extended Controller Fault Table (80h) and Read Extended I/O Fault Table (81h).	Does not support Read Extended Controller Fault Table (80h) or Read Extended I/O Fault Table (81h).	Supports Read Extended Controller Fault Table (80h) or Read Extended I/O Fault Table (81h).
SVC_REQ 23 Read Master Checksum	Returns a 15-word output block. (Same as PACSystems.)	Returns an 11-word output block. Operation is different from Series 90-70 and PACSystems. Refer to the Series 90-30 CPU Reference Manual, GFK-0467.	Returns a 15-word output block.
SVC_REQ 24 Reset Smart Module	Not supported.	Supported. (Same as PACSystems.)	Supported.
SVC_REQ 26/30 Interrogate I/O	SVC_REQ 26 is Role Switch (redundancy) service request. Interrogate I/O function is supported via fault locating references. (Same as PACSystems.)	Service Requests 26 and 30 are identical. Refer to the Series 90-30 CPU Reference Manual, GFK-0467.	SVC_REQ 26 is Role Switch (redundancy) service request. Interrogate I/O function is supported via fault locating references.

Category	Series 90-70	Series 90-30	PACSystems
SVC_REQ 36 Read from/Write to Bulk Memory Area	Supported.	Not supported.	Not supported. This service request was the way to access BMA on a 90-70 CPU. Since BMA can be accessed as %W memory in a PACSystems RX7i, the service request is no longer needed.
SVC_REQ 39 ESCM Port Status	Supported.	Not supported.	Not supported. Specific to the Series 90-70, which has an ESCM to manage its serial communications.
SVC_REQ 44 Logic Driven Dynamic Ethernet Global Data.	Supported	Not supported.	Not supported.
SVC_REQ 45 Skip Next I/O Scan	Not supported	Supported	Refer to the *Important Product Information* document for a specific CPU firmware version.
SVC_REQ 46 Fast Backplane Status Access	Not supported. (Same as PACSystems.)	Supported. Refer to the Series 90-30 CPU Reference Manual, GFK-0467 for details.	Not supported.
SVC_REQ 48 Reboot After Fatal Fault Auto Reset			
SVC_REQ 49 Auto Reset Statistics			
PID	Elapsed time computed in 10ms units.	Elapsed time computed in 10ms units.	The PID algorithm has been modified to improve some error cases; therefore PID functions differently on PACSystems. Elapsed time is computed in 100µs units instead of 10ms units. This smoothes the output characteristic, eliminating periodic adjustments that occurred when the remainder accumulated to 10ms. See chapter 10 for additional details.
Timed Contacts	Timed contact %S references are reset on a Stop to Run transition. Forces a transition if sweep is longer than half the timed contact clock cycle. Not updated in Stop mode.	Timed contact %S references are reset on a Stop to Run transition. Forces a transition if sweep is longer than half the timed contact clock cycle. Not updated in Stop mode.	Determines the state of each timed contact reference based on a free running timer that has no relationship to the start of each sweep. If the sweep time remains in phase with the timed contact clock, the contact will always appear to be in the same state. For example, if the CPU is in constant sweep mode with a sweep time setting of 100ms, the T_10MS and the T_100MS bits will never toggle. (Same as Series 90-30.) Updated in Stop mode.
Timer functions	A timer function starts accumulating time when the block is called.	Operates differently from timers in PACSystems and Series 90-70.	Same as Series 90-70. A timer function starts accumulating time on a Stop to Run transition. To prevent a timer from accumulating time, use the block fst_exe contact to reset the timers in the first sweep the block is called.

Category	Series 90-70	Series 90-30	PACSystems
OFDT function blocks	If the current value (CV) is greater than the preset value (PV) for one sweep, the CV is clamped at the PV until the function block is reset.	If the current value (CV) is greater than the preset value (PV) for one sweep, the CV remains at the higher value. It is not clamped at the PV.	Same as Series 90-70.
VME_ Functions	Supported.	Not supported.	Not supported. Replaced by BUS_ functions.

Floating Point Functions

PACSystems CPUs may return slightly different values for Not A Number (NaN) as compared to Series 90-70, Series 90-30, and VersaMax CPUs. For details on NaN values returned by floating point functions, refer to “Errors in Floating Point Numbers and Operations” in chapter 7.

Floating point functions handle the NaN propagation and cases differently. PACSystems CPUs allow the floating point hardware to handle the NaN cases, instead of treating these instructions as special cases. Impacts: ADD, SUB, MUL, DIV, SIN, COS, TAN, ASIN, ACOS, ATAN, LOG, LN, EXP, EXPT, DEG_TO_RAD, RAD_TO_DEG, ABS_REAL, SQRT_REAL functions.

PACSystems calculates TRIG functions for a larger range of values than the other PLCs. As a result, some values that previously returned NAN return a correct value.

PACSystems returns values for some unusual cases that were previously returned as NaN. Specifically 0^0 returns 1 instead of NaN.

Online Editing Mode

Category	Series 90-70	Series 90-30	PACSystems
Online editing and online testing features	Not supported	Not supported	Allow editing and testing of logic changes that are permitted for a Run Mode Store. For details on using this feature, refer to the programming software online help and *Proficy™ Logic Developer – PLC Getting Started,* GFK-1918.

Variables

Category	Series 90-70/90-30	PACSystems
Multi-bit variables mapped to discrete memory	8-bit and 16-bit variables	8-bit, 16-bit, and 32-bit variables
	Mappable to %I, %Q, %M, %T, and %G. WORD variables can sometimes be mapped to %S, depending on the instructions.	Mappable to %I, %Q, %M, %T, and %G. WORD variables can sometimes be mapped to %S, while DWORD variables can sometimes be mapped to %SA, %SB, and %SC, depending on the instructions.
Bit addressing in non-discrete memory	Not supported	You can address individual bits in BYTE, WORD, INT, UINT, DINT, and DWORD variables in non-discrete memory (%R, %AI, %AQ, %L, %P, and %W).
BOOL arrays used to replace other data types	Not supported	Supported
Index of indirect references	16 bits	32 bits when referring to %W memory. 16 bits when referring to other memory areas.
Symbolic variables	Not supported	Supported. A symbolic variable is a variable in logic that does not have an assigned reference address. Machine Edition handles all the mapping in a special portion of PACSystems user memory outside %R, %AI, %AQ, %P, %L, %W, %I, %Q, %M, %T, %S, and %G memory.
Publishing variables	Not supported	Supported

System Variables

Category	Series 90-70	Series 90-30	PACSystems
CPU Overtemperature Status (#OVR_TMP)	Not supported	Not supported	Supported
Fault Locating System Variables	Eight characters long	Not supported.	10 characters long
	Can locate 10 slots, from #0 through #9		Can locate 32 slots, from #0 through #31
	Can locate 32 modules, from #0 through #31		Can locate 256 modules, from #0 through #255

Communications

Category	Series 90-70	Series 90-30	PACSystems
Communicating with Machine Edition	Modem, serial port, and Ethernet.	Same as Series 90-70.	Ethernet supported on all versions. Serial port supported on later versions. refer to the *Important Product Information* document for a specific CPU firmware version.
Ethernet adapters	Ethernet Interface Module IC697CMM742, in any rack.	CPUs 364 and 374 have an embedded Ethernet interface. Ethernet Interface Module IC693CMM321, in any rack.	RX7i CPU modules have an embedded Ethernet daughterboard. Ethernet module IC698ETM001 in RX7i main rack only. Maximum number: 3. Ethernet module IC695ETM001 in RX3i main rack only. Maximum number: 4.
Configuring Ethernet	Involves temporarily connecting your computer to the PLC by serial cable.	Same as Series 90-70.	Can use the Set Temporary IP Address utility for a temporary connection, during which the permanent IP address can be set. Later versions support serial connection. Refer to the *Important Product Information* document for a specific CPU firmware version.
Web-based data monitoring	Not supported.	Not supported.	Up to 16 web server and FTP connections (combined)
Network routing	Supported through CMM742 Ethernet Interface configuration.	Not supported.	Not supported.
Serial ports	Can be used to communicate with Machine Edition. Provide SNP, Disabled, and Custom modes. Refer to the Serial Communications User's Manual, GFK-0582.	Same as Series 90-70.	Ports 1 and 2 provide serial interfaces to external devices. Port 1 is also used for firmware upgrades. The third on-board serial port is used as the Ethernet station manager port. . Provide RTU Slave, Message, and SNP Slave, Serial I/O, and Available modes.
Serial port default protocol	SNP	SNP	Modbus RTU.
Serial port communications from C applications	Scanf and Printf.	Same as Series 90-70.	ANSI-style read/write.

Ethernet Global Data

Category	Series 90-70	Series 90-30	PACSystems
EGD variables	Limit of 1,200 variables per exchange	Limit of 1,200 variables per exchange	No limit, as long as the total size of the EGD configuration does not exceed 64KB.
EGD upload	Does not support upload of EGD Configuration from the PLC to the Programmer	Same as Series 90-70.	Supported
Broadcast IP	Not supported	Not Supported	Broadcast option provides support for the production and consumption of EGD pages using the broadcast address of the local subnetwork.
Name Resolution	Supported.	Supported. Same restrictions as Series 90-70.	Not Supported. Adapter name in an EGD page configuration defaults to the rack.slot location.
Consumed Period	Configurable.	Configurable for CPU364. CPU374 same as PACSystems.;	Has a constant, read-only value of 200 ms.
Selective consumption	Not supported.	Not supported.	Supported. Ranges in a consumed EGD page can be set to Ignore.
%W	N/A	N/A	Supported in EGD page configuration

Flash

Category	Series 90-70	Series 90-30	PACSystems
Clearing Flash	Not supported.	Same as Series 90-70.	Supported. Clear affects all items (HWC, logic, and initial/forced values).
Reading/Verifying logic	Can read and verify logic and HWC separately.	Same as Series 90-70.	Cannot read and verify logic and HWC separately.
If EGD is present	Cannot read, write, or verify Flash memory.	Can read, write, verify, or clear Flash memory. (Same as PACSystems.)	Can read, write, verify, or clear Flash memory.
Restoring from Flash	CPX models same as 90-30. NA for other 90-70 models.	When powering up from flash, the 90-30 CPU powers up in the mode that it was in when it powered down. If the battery-backed memory is corrupt (no battery, super capacitor discharged) the 90-30 CPU can be configured to power up from flash.	OEMs can configure PACSystems to automatically restore user programs/data from Flash to battery-backed RAM. After such a restoration takes place, the PLC boots from RAM and not Flash, as long as the RAM's contents are valid (as determined during the power-on tests). For a detailed explanation, see "Logic/Configuration Source and CPU Operating Mode at Power-up" in Chapter 5.
Writing to Flash	Does not write the transitions.	Does not write the transitions.	Takes a snapshot of the transitions that are currently set and writes them to Flash

Category	Series 90-70	Series 90-30	PACSystems
Verifying Flash	Does not verify transitions.	Does not verify transitions.	Verifies the transitions. If a transitional changed value from 0 to 1 and back to 0, the value would be equal, but the transition could be unequal.
Store/Restore transition bits from flash	When reference tables are restored from flash, transition bits are set according to the differences between what was in RAM versus what was in flash at the time of the restore.		Stores/restores the state of the transition bits to/from flash along with the status values and overrides of each reference table.
Logic/Configuration source and CPU Mode when Flash contains no configuration and Powerup Source in RAM is Always Flash		Uses default logic/configuration and goes to Stop Disabled mode.	Uses logic/configuration from RAM. CPU mode determined as described in "Effect of Logic/Configuration Power-up Source on CPU Behavior," chapter 5.
Interruptible Flash Read/Write	The entire transfer process must complete before it can be canceled.	The entire transfer process must complete before it can be canceled.	The contents of flash memory or RAM are copied as individual files. This allows you to cancel a flash read or write operation during the copy process instead of waiting for the entire transfer process to complete.

Memory

Differences in the Memory Areas Supported

Memory Area	Series 90-70	Series 90-30	PACSystems
%GA through %GE	Supported	Uses %G (same as PACSystems).	Uses %G instead. Target conversion from Series 90-70 PLC to PACSystems automatically converts %GA - %GE memory mappings to %G mappings. For details, see "Changes Made During the Conversion," page C-25.
Bulk Memory Area	Accessed by means of SVC_REQ 36	N/A	Accessed by mapping variables to %W memory
%W	In LD programs, the BMA is accessed by the SVCREQ 36 function. In C programs, the BMA is accessed through the PLCC_buil_mem() function.	N/A	Supported. Represents the Bulk Memory Area.
Symbolic	Not supported	Same as Series 90-70.	Supported. One memory area is used for discrete symbolic variables and another for non-discrete symbolic variables.

Maximum Memory Sizes

The maximum memory size of some memory areas in PACSystems targets is larger than in Series 90 PLCs.

Memory Area	Series 90-70	Series 90-30	PACSystems
%AI, %AQ	8,192 words (16,384 bytes)	32640	32,640 words (65,280 bytes) each
%I, %M, %Q	12,288 points (1,536 bytes)	2048	32,768 points (4,096 bytes) each
%T	256 points (32 bytes)	256	1,024 points (128 bytes)
%R	16,384 words (32,768 bytes)	32640	32,640 words (65280 bytes)
%W	Not supported	Not supported.	Maximum available user RAM
Total user space			10 megabytes

For details on items that count against user memory, refer to appendix B.

Other Differences in Memory Support

Memory Area	Series 90-70	Series 90-30	PACSystems
%P and %L	Buffer size not configurable.	Not supported.	Buffer size can be configured through the LD block's Extra Local Words property

Redundancy

Category	Series 90-70	Series 90-30/RX3i	PACSystems Rx7i
CPU redundancy	Supported	Not supported.	Supported in redundancy CPUs
Genius redundancy	Supported		Supported. CPU over Genius supported in redundancy CPUs.
IP Address redundancy	Supported		Supported in redundancy CPUs.

Genius

Category	Series 90-70	Series 90-30	RX3i	PACSystems RX7i
Handling loss of GBC	Sets input data for devices associated with a failed GBC to Hold Last State, regardless of how the Input Default parameter is configured for each device.	Sets input data for devices associated with a failed GBC to zero.	Sets input data for devices associated with a failed GBC to the state specified (Default/Hold Last State) by the Input Default parameter for the GBC.	Sets input data for devices associated with a failed GBC to the state specified by the Input Default parameter for each device.
Handling losses and additions of Genius device	The Series 90-70 GBC logs a Loss of Device or an Addition of Device fault and the Series 90-70 PLC updates the appropriate fault locating reference.	The Series 90-30 GBC does not log a fault. The Series 90-30 PLC does not support fault locating references.	The Series 90-30 GBC does not log a fault, therefore the RX3i does not update fault locating references.	The Series 90-70 GBC logs a Loss of Device or an Addition of Device fault and the RX7i updates the appropriate fault locating reference.
Handling of data for lost redundant Genius blocks	Immediately applies the default input data to the input reference tables and updates the associated diagnostic tables.	N/A	NA	Updates input data and input diagnostic tables with the default data during the input scan immediately following the loss of device fault. Updates output diagnostic data tables during the output scan immediately after the loss of device fault.
Setting of Force Present status bit, FRC_PRE %S12.	Limited to Genius blocks.	NA	Not limited to Genius blocks. If any input module reports that it has a force present, this bit is set.	Not limited to Genius blocks. If any input module reports that it has a force present, this bit is set.
Point fault references	Supported by Series 90-70 CPUs, Series 90-70 GBC, and other legacy Series 90-70 modules.	Not supported by Series 90-30 CPU or legacy Series 90-30 modules.	RX3i CPU supports point fault references. Series 90-30/RX3i GBC does not.	Supported

I/O and Intelligent Modules

Category	Series 90-70	Series 90-30	PACSystems
Discrete output modules	Discrete output modules controlled by a Series 90-70 CPU drive their outputs only after the second sweep. DO I/O has no effect in the first sweep because outputs are not enabled until they are part of an output scan.	Same as PACSystems	Discrete output modules controlled by a PACSystems CPU drive their outputs to the values set by user logic the first time they are part of an output scan. A DO I/O function causes a module's outputs to be enabled at the point of the DO I/O execution (if that module's outputs have not already been enabled).
Default conditions for input modules on STOP to RUN transition.	When the CPU transitions from STOP to RUN mode, the modules' input values are retained.		When the CPU transitions from STOP to RUN mode, the modules' input values are set to 0.
Do I/O to High Speed Counter	If %AI is specified, returns only AI input data. If %I is specified, returns all input data (I and AI).	Same as PACSystems	Returns all input data (I and AI).
I/O interrupt blocks	Prioritizes timed interrupts before I/O interrupts.	NA (does not support I/O interrupts)	Normally scheduled relative to each other on a first-come first-served basis. CPUs with later firmware versions allow preemptive block scheduling. For availability on a given CPU version, refer to the *IPI* document provided with that CPU.
Setting up an I/O Interrupt Block for an I/O Address that is not configured and/or enabled for interrupts.	No faults - CPU can go to Run mode.	NA (does not support I/O interrupts)	CPU goes to Stop-Fault mode after download to CPU. Controller Fault Table must be cleared to place CPU in Run mode. Module interrupts should be configured/enabled in hardware configuration.
I/O interrupts on High Speed Counter and IC697MDL641 module	The legacy transitions (POSCON/NEGCON) are not set.	N/A	The legacy transitions are always set for every one of the discrete input references at the time the interrupt occurs. (If the new state of the bit is on, the up transition contact will be true. If the new state is OFF, the down transition contact will be true).

Category	*Series 90-70*	*Series 90-30*	*PACSystems*
Analog Input Interrupts	HIALR and LOALR fault contacts are not set at the time the interrupt logic is run.	Point faults not supported.	If point faults are enabled, the HIALM or LOALM fault contact will be set to TRUE before running the interrupt logic.
Analog Expansion modules	No limit on number of expanders. Analog base module and expansion modules can be assigned to different scan sets.	N/A	(RX7i only) Maximum of three expanders allowed in an RX7i rack. Expansion modules are assigned to the same scan set as the analog base module.

Converting an Application from Series 90-70 to PACSystems

Warning

There may be execution differences when converting an application from a Series 90-70 target to a PACSystems target. It is the application developer's responsibility to validate and test the application execution prior to deployment into a production environment.

Caution

GE Fanuc PLC target conversions are irreversible. When logic blocks are deleted during a conversion, they cannot be restored. That is, there is no undo to a conversion. It is recommended that you make a backup of your project before converting a target in it.

Note: If a Series 90-70 target contains State Logic, it cannot be converted to PACSystems.

Preparing for the Conversion

Analog Expander Modules

If you want to convert a Series 90-70 rack configuration that contains IC697ALG440 or IC697ALG441 analog input expander modules, you must ensure that the IC697ALG230 base module and the expander modules are located in certain slots as detailed below; otherwise, the modules will not be converted. (If you do not use expander modules, the base module can be configured in any slot.) If you use expander modules, you must put the base module in slot 3, 4, or 5 of the Series 90-70 rack before conversion.

- If you put the base module in slot 3 of the Series 90-70 rack, you must put the IC697ALG440 or IC697ALG441 expander modules in slots 4, 5, or 6.
- If you put the base module in slot 4, the expander modules can go in slots 5 or 6.
- If you put the base module in slot 5, you can have only one expander module, in slot 6.

Note: Another convertible configuration is a base module in slot 3 and its expander module in slot 4, and another base module in slot 5 and its expander module in slot 6.

PCM, CMM, and DLAN Modules

If you want to convert a Series 90-70 rack configuration that contains a IC697PCM711, IC697CMM711, or IC697BEM761 you must ensure that these modules are located in slots less than or equal to 9 in an RX7i rack. If any of these modules occupies a slot greater than 5 in the 90-70 rack configuration, it will not be converted because the corresponding destination slot in the RX7i rack configuration would be greater than 9. For example a IC697PCM711 module in slot 6 of a Series 90-70 rack configuration would not be converted because the destination in the RX7i rack would be slot 11.

Note: If your 90-70 application uses a PCM, CMM, or DLAN module, please refer to "Series 90-70 Communications and Intelligent Option Modules" of Chapter 11 for important information about using these modules with PACSystems RX7i.

VME_ Instructions

The PACSystems Rx7i provides Bus_Read and Bus_Write functions similar to the Series 90-70 VME_Read and VME_Write functions. When converting a Series 90-70 application, some modifications to hardware configuration and logic are required to use the new instructions. These changes must be made manually after converting the application. Before converting your application, review the descriptions of VME addressing and information needed to complete the conversion.

Note: For details on selecting, configuring, and programming non-GE Fanuc VME modules in a PACSystems control system, refer to *PACSystems RX7i User's Guide to Integration of VME Modules,* GFK-2235.

PACSystems vs. Series 90-70 VME Addressing Schemes

Non-GE Fanuc VME Modules

A typical VME board is configured to respond to certain VME memory addresses. Most non-GE Fanuc boards are configured using one or more jumpers to set an Address Modifier (AM code) and a base address that the board will respond to. Typically a board has one or two specific memory areas that can be accessed. PACSystems refers to these areas as *Regions*. A Region is comprised of a base address, AM code, and size, as well as a “width”, termed *Interface Type*, (byte, word, dword, etc) that can be accessed.

In PACSystems, each VME *Region* on a board is specified in the Hardware Configuration. PACSystems supports a maximum of eight regions per module. A Region specification includes a region number, AM code, Base Address, Size, and Interface Type. Specifics of these settings are covered in the programming software Help Information. The PACSystems BUS_ functions include a region (RGN) parameter that refers to the regions configured in the Hardware Configuration information.

In Series 90-70, the AM code and VME address are specified as parameters to the VME_ function itself. The “width” specification is determined by the instruction used (for example, READ_BYTE versus READ_WORD).

Information Needed to Complete the Conversion

To complete the conversion process, you will need to identify the memory area(s) being used on each VME module, including AM code, and VME base address. This information should be provided with the VME board being used. If not available, this information can be obtained by reviewing the logic in the Series 90-70 application. As a general rule, a separate region will be needed for each AM code used.

PCM Applications

The VME address assignments for VME modules in an RX7i system differ from the assignments in a 90-70 system. If an application program running on the IC697PCM711 accesses the VME bus (that is, it uses Set_vme_ctl, Vme_read, Vme_test_and_set, or Vme_write), the VME addresses used by that program will need to be updated to be in agreement with the PACSystems RX7i VME addressing assignments. To determine the correct VME addresses to use on the RX7i, please refer to the following sections in the *PACSystems RX7i User's Guide to Integration of VME Modules,* GFK-2235:

"VME Addresses for GE Fanuc Modules in the Main Rack"
"VME Addresses for GE Fanuc Modules in Expansion Racks"

Also please note that the S9070_xxxx macros, listed below, that are provided by the PCM C toolkit in the file Vme.h ***cannot*** be used to calculate VME addresses in an RX7i system.

```
S9070_RACKSLOT_VALID(r,s) (r>=0&&r<=7&&s>=2&&s<=9)

S9070_VME_HI_ADDR(r,s) ((r)?(0xF0-(0x10*(r))+2*((s)-2)):(2*((s)-2)))

S9070_VME_SHORT_ADDR(s) (0x800*s)
```

Converting the Target

To convert a Series 90-70 target to a PACSystems RX7i target in the programming software:

1. In the Project tab of the Navigator, right click the target you want to convert and select Properties. The Inspector displays the target properties.
2. In the Properties, select the PACSystems RX7i family. A Target Conversion Warning message appears. If you want to continue with the conversion, click OK.
3. The target is converted to the PACSystems RX7i family. The target conversion report is displayed in the InfoViewer when the conversion is complete.
4. Review the target conversion report, correct any problems identified, and validate the application. The application should be thoroughly tested to detect problems that may be caused by execution differences before deploying it in a production environment.

   For additional information, please refer to "Changes Made During the Conversion," page C-25 and "Finishing the Conversion," page C-26.

Warning

There may be execution differences when converting an application from a Series 90-70 target to a PACSystems target. It is the application developer's responsibility to validate and test the application execution prior to deployment into a production environment.

Changes Made During the 90-70 to PACSystems RX7i Conversion

Each Series 90-70 module supported by a PACSystems RX7i target is remapped from its Series 90-70 rack and double-width slot to the corresponding PACSystems RX7i rack and single-width slots.

Note: Because a PACSystems RX7i power supply uses only one slot, the resulting PACSystems RX7i slot number is calculated as follows:

PACSystems RX7i slot = (2 * [Series 90-70 slot number]) - 1

- The parameter values for each converted module are preserved whenever possible. Parameters unique to RX7i are set to their default settings.
- Ethernet Global Data (EGD) is converted. For each rack, adapter 1 is converted to the RX7i's CPU embedded Ethernet daughterboard, while adapters 2, 3, and 4 are converted to RX7i Ethernet modules (IC698ETM001), and adapters 5 and beyond are ignored.

Note: The Series 90-70 slot in which adapter 1 resides is left empty in RX7i, as per the formula empty RX7i slot = (2 * [Series 90-70 slot number of adapter 1]) - 1

- References to %GA - %GE memory areas are converted to %G addresses with new offsets as follows:

Preconversion memory type	*Postconversion memory type and offset*
%G	%G+0
%GA	%G+1280
%GB	%G+2560
%GC	%G+3840
%GD	%G+5120
%GE	%G+6400

- C blocks are retained and flagged in the report. You may need to edit them. You will also need to recompile them with the PACSystems C Toolkit and update them in the PACSystems target.
- All C programs are deleted.
- LD blocks are converted and scanned for instructions that require updating.

Note: The first validation of a converted target flags the LD instructions that are not supported in the RX7i.

- The original Series 90-70 system variables are deleted except when they are used in logic. PACSystems RX7i system variables are added to the target. A warning is reported for each system variable found to be used in logic. It is up to you to ensure that these system variables are still valid for the new target type.
- When you convert a Series 90-70 target to an RX7i target, all Series 90-70 fault locating system variables are converted to PACSystems RX7i versions by inserting a 0 before the slot number. For example, if #BUS_121 was used on a Series 90-70 target, the variable is renamed to #BUS_1021 when the target is converted to RX7i. Also, the #RACK_0r variables are converted to #RACK_00r.

Finishing the Conversion

Review the Target Conversion Report

The target conversion report, displayed when the conversion is completed, summarizes the results of hardware configuration conversion and logic conversion. Items that were not converted are identified.

Note: The conversion report does not warn about all possible differences in logic execution. A validity check after conversion may report problems that could not be detected during conversion. Execution differences may exist when converting from Series 90-70 to PACSystems RX7i, even for rungs that were not mentioned in the report or reported during validity checks.

The report provides an analysis of each LD block that warns of unsupported instructions, unsupported service requests, fault locating reference usage, instructions that were converted, and instructions that could not be converted and why.

Lines displayed in red characters warn of the most important potential differences in logic execution (for example, due to the different execution order of a Series 90-70 PLC or PACSystems RX7i when faced with branches in logic). For each potential difference reported, you should examine the logic. Look for variables used as both inputs and outputs on different instructions where the instruction with the input is not on the same row as the instruction with the output.

The report is saved in the Documentation files folder in the Supplemental Files folder in the resulting PACSystems RX7i target. You can print the report directly from the InfoViewer or print it from the copy saved in the Documentation Files folder.

Replace VME_ Instructions with BUS_ Instructions

RX7i provides a set of BUS_ instructions that can be used to access data in a module on a bus: BUS_RD, BUS_WRT, BUS_RMW, and BUS_TS. These instructions are similar to the VME_ instructions supported by Series 90-70. Series 90-70 VME_ instructions are ***not*** automatically converted to RX7i BUS_ instructions.

The following is a summary of the changes that must be made after converting the application from Series 90-70 to PACSystems.

- Determine the number of regions needed for a given board, and the AM code and base address for each region. Note that each AM code requires a separate region.
- Using Machine Edition hardware configuration, configure each non-GE Fanuc board, adding the appropriate number of regions, and specifying the necessary AM Code, Base Address, Size, and Interface type. Each region will be referred to by number within the LD program.
- Modify the LD function calls.
  - Replace each VME_ instruction with the corresponding BUS_ instruction. (For example, VME_READ_WORD should be replaced by BUS_READ_WORD; VME_CFG_READ should be replaced by a BUS_READ_ instruction, etc.) Unsupported instructions are identified in the Target Conversion Report.
  - Add the Rack and Slot parameters to refer to the appropriate non GE Fanuc VME module.

- Specify the appropriate Region number used in the hardware configuration.
- Compute the “address” parameter for the function. Note that the value needed is now an ***offset*** relative to the specified region as opposed to an absolute VME address. Therefore, you must ***subtract*** the base address specified by the region you are referring to. For example, if the Series 90-70 instruction used an address of 0x400100 and you have specified 0x400000 as the base address of the region, you would enter 0x100 as the offset.
- If your RX7i application program needs to access the dual port memory of a PCM, CMM, or DLAN, use the BUS READ and WRITE functions. When accessing one of these modules, set the function’s Region parameter to 1. (For the PCM, CMM, and DLAN modules, region 1 is predefined to be the module's entire dual port memory. Configure these modules according to their catalog numbers; do not configure these modules as “VME modules.”)

Note that other (optional) parameters have been added. Specifics on these parameters are provided in the Machine Edition online help.

Increasing Stack Allocation for Programs Converted from Series 90-70

Series 90-70 programs are converted to RX7i with the same stack allocation. RX7i uses more stack space than the Series 90-70, so some user programs may not run after conversion. To increase the stack space, right click the _MAIN block and select Properties. Stack Size is listed at the bottom of the Properties page. The default stack size in RX7i programs is 64KB. Programs with a large number of nested calls may need more stack space. As a general rule, the stack for the converted RX7i program should be set to approximately three times the stack size of the Series 90-70 version of the program. A diagnostic fault is displayed if the program runs out of stack space.

Converting an Application from Series 90-30 to PACSystems

Preparing for the Conversion

End Instruction

PACSystems does not support the END instruction, which unconditionally terminates program execution and transfers control to the beginning of the program (the first rung of the _MAIN block) for the next scan. Following are suggestions for reorganizing programs that use the END instruction:

- Adding JUMP(s) to the end of the block(s) in the CALL chain
- Separating the logic following the END instruction into a separate block that is called only if the END would not have been executed.
- Implementing other means of debugging the LD program, such as saving copies of temporary register contents when the circumstances being debugged occur.

Updated 24V Analog Modules (IC693ALG220, IC693ALG221, and IC693ALG222C)

If you are using an old version (ALG220G, ALG221G, ALG222C or earlier) of these modules, you must provide an external 24V power source to the backplane terminals, or replace the old version with the latest version.

SRTP Communication with Older Clients

The RX3i CPU can be placed in any slot in rack 0 (except the last slot). PACSystems provides an SRTP Destinations service that allows an external client using SRTP to find the CPU's rack and slot location in order to communicate with it. However, legacy clients, such as Series 90 PLC applications using SRTP channels and HCT host applications do not use this service. They assume that the server CPU is always in rack 0, slot 1 as is required in the Series 90 systems.

To support communications with legacy SRTP clients such as Series 90 PLCs using SRTP Channels, the RX3i redirects service requests arriving on an SRTP server connection destined for rack 0 slot 1 to rack 0 slot 2 if:

- There is only one CPU in the system.
- The CPU is located in rack 0 slot 2.
- The remote client has not issued an SRTP Destination service on the connection to discover the rack and slot of the CPU.

Redirecting the services from rack 0 slot 1 to rack 0 slot 2 consists ***only*** of changing the rack and slot portion of the destination address within the service request mailbox message. The content (payload) of the service request is not examined or modified, nor is the service request response mailbox message from the CPU.

All services used by SRTP channels clients in the Series 90 and all HCT services can be successfully redirected.

CPU Slot Location

The RX3i CPU is a double-width module whose connector is right justified as viewed when installed in a rack. It is referenced for configuration and by user logic applications by the leftmost slot that it occupies. For example, if the RX3i CPU has its physical connector inserted into slot 4, which means it occupies slots 3 and 4, the CPU is referenced as being located in slot 3. The referenced location of the CPU is not determined by what slot the physical connector is located in, but by the leftmost slot occupied by the entire module.

The RX3i CPU may be located in any slot in the main rack except slot 11 of a 12-slot rack or slot 15 of a 16-slot rack, because these slots would require the physical connector to be located in the slot reserved for an expansion module.

When migrating a Series 90-30 CPU system to a PACSystems RX3i CPU, be aware that to maintain the Slot 1 location of the CPU, only a single-width power-supply may be used in slot 0. Therefore, if the application using an existing Series 90-30 system must maintain a slot 1 CPU and uses a double-width power-supply, the power supply must be located in a slot to the right of the RX3i CPU in Slot 1.

In deciding to place the CPU in a slot other than Slot 1, you should be aware of the possible application migration issues that could arise. The following table lists the areas that could be affected when migrating an application from one CPU slot to another.

CPU Slot Placement Issues

Item Affected		***How Affected***
User Logic	Service Request #15 (Read Last-Logged Fault Table Entry)	Location of CPU faults will not be the standard 0.1 location, but will reflect the slot the CPU is located in. User logic that decodes fault table entries retrieved by these service requests may need updating.
	Service Request #20 (Read Fault Tables)	
	Communications Request (COMM_REQ)	COMM_REQs directed to the CPU (e.g. those directed to the serial ports of the CPU) will need to be updated with the correct CPU slot reference.
H/W Configuration	CPU Slot location	Slot location of the CPU must be updated in the HW Configuration to reflect the CPU's true location.
Fault Tables	Faults logged for the CPU	The location of faults logged for the CPU in the fault table will not be the standard 0.1 (rack.slot) location, but will reflect the CPU's actual slot.
External Devices	*Series 90 PLCs*	
	Remote Series 90 PLCs that use SRTP Channels COMM_REQs expect the CPU to be in slot 1. To support communications with Series 90 SRTP clients such as Series 90 PLCs using SRTP Channels, the RX3i internally redirects incoming SRTP requests destined for {rack 0, slot 1} to {rack 0, slot 2}, provided that the CPU is located in rack 0 slot 2 (and the remote client has not issued an SRTP Destination service on the connection to discover the rack and slot of the CPU). This special redirection permits Series 90-30 applications that expect the power supply to be located leftmost and the CPU to be located to the right of the power supply to function. Attempts to establish channels with CPUs in slots other than 1 or 2 will fail if initiated from Series 90 PLCs.	
	HMI and External Communication Devices	
	All external communication devices that interact with the CPU should be checked for compatibility with CPU slot locations other than slot 1. Problems may arise with, but are not limited to, initial connection sequences and fault reporting. View -Machine Edition customers should select "GE SRTP" as their communications driver – it can communicate with a CPU in any slot.	
	Host Communications Toolkit (HCT)	
	Applications that utilize the Host Communications Toolkit may require updated drivers.	

Converting the Target

To convert a Series 90-30 target to a PACSystems RX3i target in the programming software:

1. In the Project tab of the Navigator, right click the target you want to convert and select Properties. The Inspector displays the target properties.
2. In the Properties, select the PACSystems RX3i family. A Target Conversion Warning message appears. If you want to continue with the conversion, click OK.
3. The target is converted to the PACSystems RX3i family. The target conversion report is displayed in the InfoViewer when the conversion is complete.
4. Review the target conversion report, correct any problems identified, and validate the application. The application should be thoroughly tested to detect problems that may be caused by execution differences before deploying it in a production environment.

   For additional information, please refer to “Changes Made During the Conversion,” page C-31 and “Finishing the Conversion,” page C-32.

Warning

There may be execution differences when converting an application from a Series 90-30 target to a PACSystems target. It is the application developer's responsibility to validate and test the application execution prior to deployment into a production environment.

Changes Made During the 90-30 to PACSystems RX3i Conversion

Hardware Configuration

- Each Series 90-30 module supported by a PACSystems RX3i target is remapped from its Series 90-30 rack slot to the corresponding PACSystems RX3i rack and slots.
- The RX3i CPU and default power supply require two slots. Slot locations of other modules are adjusted as needed.
- Series 90-30 CPUs with embedded Ethernet interface are converted to an RX3i CPU and an IC695ETM001 peripheral Ethernet module. Slot locations are adjusted as needed.
- The parameter values for each converted module are preserved whenever possible. Parameters unique to RX3i are set to their default settings.
- Power consumption requirements are converted from Watts to Amps.

Logic

- C blocks are retained and flagged in the report. You may need to edit them. You will also need to recompile them with the PACSystems C Toolkit and update them in the PACSystems target.
- All C programs are deleted.
- IL (Instruction List) and SFC (Sequential Function Chart) programs are not translated. IL and SFC programming are not supported.
- LD blocks are converted and scanned for instructions that require updating.

The following instructions are flagged as not supported:

- SVC_REQ #41 (PEEK), SVC_REQ #42 (Daughterboard Info). These are not translated. Other means of debugging the operation of the system and of determining daughterboard revision information are provided.
- SVC_REQ 46 Fast Backplane Status Access
- SCV_REQ #48 & #49 (auto-restart parameters). These are not translated, since the auto-restart feature is not implemented. The program is translated successfully without them, but you are notified that they have been omitted.
- END. Not supported.

The following instructions are flagged for manual translation:

- SVC_REQ 6, Change/Read Number of Words to Checksum
- SVC_REQ 15, Read Last-Logged Fault Table Entry
- SVC_REQ 23, Read Master Checksum
- SVC_REQ 26/30, Interrogate I/O. Note that the Series 90-30 Interrogate I/O functionality is supported in PACSystems by fault locating references.

The following instructions are changed:

- WORD_TO_REAL instruction translated to UINT_TO_REAL.
- REAL_TO_WORD instruction translated to REAL_TO_UINT.
- Enhanced DO_IO translated to standard DO_IO (The constant ALT parameter is discarded and ignored.)
- Non-nested JUMP, LABEL, MCR, & ENDMCR. These are translated to the corresponding nested JUMPs, LABELs, MCRs, & ENDMCRs.

Finishing the Conversion

Review the Target Conversion Report

The target conversion report, displayed when the conversion is completed, summarizes the results of hardware configuration conversion and logic conversion. Items that were not converted are identified.

Note: The conversion report does not warn about all possible differences in logic execution. A validity check after conversion may report problems that could not be detected during conversion. Execution differences may exist when converting from Series 90-30 to PACSystems, even for rungs that were not mentioned in the report or reported during validity checks.

The report provides an analysis of each LD block that warns of unsupported instructions, unsupported service requests, fault locating reference usage, instructions that were converted, and instructions that could not be converted and why.

Lines displayed in red characters warn of the most important potential differences in logic execution. For each potential difference reported, you should examine the logic.

The report is saved in the Documentation files folder in the Supplemental Files folder in the resulting PACSystems target. You can print the report directly from the InfoViewer or print it from the copy saved in the Documentation Files folder.

@

@
  indirect references, 7-6

A

Absolute Value, 8-116
Add, 8-117, 9-23
Addition of I/O module, 15-53
Addition of IOC, 15-51
Addition of or extra rack, 15-16
Address operators, 12-2
Advanced math functions, 8-2, 9-2
Advanced user parameters, 4-6
Alarm contacts, 15-13
Analog expander modules
  fault locating references, 15-12
Analog I/O diagnostic information, 5-25
Analog input register references (%AI), 7-6
Analog output register references (%AQ), 7-6
Application fault, 15-29
Applications, converting, C-22
Array Move, 8-99
Auto-Located symbolic variables, 7-2

B

Base sweep time, A-12
  status faults, 15-28
Battery life
  CPE010/CPE020/CRE020, 2-4
  CPE030/CRE030, 2-7
  CPE040/CRE040, 2-7
  CPU310, 2-13, 2-15
Battery status fault, 15-28
Baud rates, serial port, 13-7
Bit in Word references, 7-7
Bit Operation Functions, 8-7, 9-4
Bit Position, 8-9
Bit references, 7-8
Bit Sequencer, 8-10
Bit Set, Clear, 8-13
Bit Test, 8-14
Block Clear, 8-76
Block Move, 8-77
Block switch, 15-56
Blocks
  external, 6-10
  parameterized, 6-4
  program, 6-3
  types of, 6-3
  UDFBs, 6-6
Boolean execution times, A-1
  RX3i, 2-13, 2-15
  RX7i, 2-3, 2-6
Bulk memory, 7-6
BUS_ functions, 8-78

C

Cables, 13-7
Call, 8-127
Change background task window mode and timer, 10-9
Change backplane communications window mode and timer, 10-8
Change controller communications window mode, 10-7
Change/read constant sweep timer, 10-4
Change/read number of words to checksum, 10-10
Clear fault tables, 10-24
Clocks, 5-16
  elapsed time
    reading with SVCREQ #16 or #50, 5-16
  elapsed time clock, 2-3, 2-13, 2-15, 5-16
  time-of-day clock, 2-3, 2-6, 2-13, 2-15, 5-16
    reading and setting, 5-16, 10-12
    synchronizing to SNTP server, 5-17
CMM, 13-8
Coil Checking, 8-25
Coils, 8-25
Column-major logic execution, 6-14, C-3
Comment, 8-130, 9-9
Communication requests (COMM_REQ), 8-84
  serial I/O, 14-9
  SNP, 14-52
  using to configure serial ports, 14-2
Communications Coprocessor, 13-8
Communications failure during store, 15-36
Compare function, 8-136
Comparison Functions, 8-135, 9-10
Configuration
  parameters, CPU, 3-2
  parameters, embedded Ethernet Interface, 4-3
  storing (downloading), 3-17
  system, 5-27
Constant sweep time exceeded, 15-28
Contacts, 8-31
  continuation, 8-32
Control Functions, 8-39, 9-13
Convenience references. See System status references
Conversion functions, 8-58, 9-32
  BCD4, INT, DINT, or REAL to UINT, 12-7

BCD4, UINT, DINT, or REAL to INT, 12-7
BCD8, UINT, or INT to DINT, 12-7
Converting an application, C-22
Corrupted user program on power-up, 15-33
Coummunication requests
Serial IO
calling from CPU sweep, 14-7
Counters, 8-139, 9-15
CPU hardware failure, 15-24
CPU memory validation, 5-26
CPU performance data
base sweep time, A-12
Boolean execution times, A-1
calculating predicted sweep times, A-27
I/O interrupt performance and sweep impact, A-25
I/O module sweep impact times worksheet, A-18
I/O scan and I/O fault sweep impact, A-15
instruction timing, A-2
programmer sweep impact time, A-14
sweep impact of Ethernet global data, A-21
sweep impact of Genius I/O and GBCs, A-19
sweep impact of I/O modules, A-15
sweep impact of intelligent option modules, A-24
CPU redundancy, 3-10, 15-35
CPU sweep
STOP mode, 5-10
Cyclic redundancy check (CRC), 14-28

D

Data coherency in communications windows, 5-9
Data Initialize, 8-89
ASCII, 8-90
Communications Request, 8-91
DLAN, 8-91
Data mapping
default conditions, 5-22
Genius I/O data mapping, 5-23
Data Move functions, 8-75, 9-16
Data retentiveness, 7-12
Data scope, 7-13
Data Table Functions, 8-97
Data types, 7-19
Datagrams
permanent, 14-52
Determining if an IP address has been used, 4-7
Diagnostic information, analog I/O, 5-25
Diagnostic information, discrete I/O, 5-24
Disable/enable EXE block and standalone C program checksums, 10-43
Discrete references, 7-8
size and default, 7-9
Divide, 8-119, 9-24
DLAN Interface, 13-10
Do I/O, 8-40
in an interrupt block, 6-20
Documentation, 1-8
Down Counter, 8-72
Downloading configuration, 3-17
Drum, 8-45

E

Elapsed time clock, 5-16
accuracy, 2-3, 2-13, 2-15
reading with SVCREQ #16 or #50, 5-16
Equal, 8-137, 9-12
Error checking and correction (ECC)
CRE020, 2-4
CRE030, CRE040, 2-7
Errors
in floating point numbers, 7-21
Ethernet global data, 2-1
sweep impact times, A-21
timestamping, 5-17
Ethernet Interface
configuring, 4-2
embedded, 13-2
modules, 13-2
ports, 2-8, 13-2
verifying power-up, 4-7
Exponential/Logarithmic Functions, 8-3, 9-3
Expressions
Structured Text, 12-1
External blocks, 6-10
Extra block fault, 15-54
Extra I/O module, 15-53

F

Fatal faults
CPU system software failure, 15-35
Fault contacts, 8-32, 15-11
Fault handling
actions, 15-3
CPU configuration, 3-8
overview, 15-2
system, 15-8
system response, 15-2
Fault references
alarm contacts, 15-13
fault locating, 15-11
point faults, 15-13
Faults
Controller, 15-14
CPU system software failure, 15-35

fault contacts, 15-11
I/O, 15-38
system, 15-8
non-configurable, 15-10
tables, 15-4
tables, IO, 15-6
tables, PLC, 15-4
user-defined, 7-16, 10-38, 15-5
Flash memory, 5-13
Floating point numbers, 7-20
errors in, 7-21
PACSystems vs. others, C-13
For Loop, 8-48
Forced and unforced circuit, 15-52
Formal parameters
in ST calls, 12-8
restrictions, 6-5
Function Block Diagram language, 6-15, 9-1

G

Gateway address, 4-3
GBC software exception, 15-55
GBC stopped reporting, 15-54
GBC Stopped Reporting fault, 15-54
GENA (Genius Network Adapter), 15-39
Genius global data, 5-24, 7-10
Genius I/O, 5-24
analog grouped block, 5-24
default conditions, 5-23
diagnostic data collection, 5-24
Genius I/O data mapping, 5-23
low-level analog blocks, 5-24
Global data references (%G), 7-8
Greater or Equal, 8-137, 9-12
Greater Than, 8-137, 9-12

H

Hardware variables, 7-3
High and Low Alarm Contacts, 8-33

I

I/O bus faults, 15-49
I/O circuit faults, 15-41
I/O data mapping
default conditions, 5-22
Genius I/O data mapping, 5-23
I/O fault sweep impact, A-15
I/O fault table full, 15-29
I/O interrupts, 6-19
performance and sweep impact, A-25
I/O module faults, 15-50
I/O module sweep impact times
worksheet, A-18
I/O scan sets, 5-22
configuration, of, 5-22
I/O scan sweep impact, A-15
I/O system
analog I/O diagnostic information, 5-25
discrete I/O diagnostic information, 5-24
initialization, 5-27
I/O variables, 7-3
Indirect references
word, 7-6
Initialize Port function, 14-11
Input Buffer, Flush, 14-11
Input Buffer, Set Up, 14-11
Input references (%I), 7-8
Instruction set
operands
LD, 7-24
PACSystems vs. other controllers, C-9
Instruction timing, CPU, A-2
Intelligent option modules
self-test completion, 5-27
sweep impact times, A-24
Internal references (%M), 7-8
Interrupt blocks, 6-17
I/O interrupts, 6-19
interrupt handling, 6-17
module interrupts, 6-19
scheduling, 6-19
timed interrupts, 6-18
Inverse Trig Functions, 8-6
IOC (I/O controller), 15-9
IOC hardware failure, 15-54
IOC software fault, 15-52
IP address, 4-3
Configuration, 4-3
Determining if it has been used, 4-7

J

Jump, 8-131

L

Ladder Diagram language, 6-14, 8-1
Last scans, 3-5, 5-10
LDPROG01, 6-1
LEDs, 4-7
CPE010, CPE020, CRE020, 2-2
CPE030/CRE030, CPE040/CRE040, 2-5
CPU310, 2-12
CPU320, 2-14
Ethernet interface, 2-8
Less or Equal, 8-137, 9-12
Less Than, 8-137, 9-12
Logic Execution

row-major vs. column-major, 6-14, C-3
Logic/configuration power-up source, 5-14
Logical AND, OR, and XOR, 8-15, 9-6
Logical NOT, 8-18, 9-8
Loss of I/O module, 15-53
Loss of IOC, 15-51
Loss of or missing option module, 15-16
Loss of or missing rack, 15-15
LREAL numbers
internal format of, 7-21

M

Mapping, I/O data
default conditions, 5-22
Genius I/O data mapping, 5-23
Mask I/O Interrupt, 8-51
Mask/unmask IO interrupt, 10-29
Mask/unmask timed interrupts, 10-40
Masked Compare, 8-19
Master Control Relay/End Master Control Relay, 8-132
Math functions, 9-21
advanced, 8-2, 9-2
Math Functions, 8-115
Memory
configuration, 3-6
retention of data memory across power failure, 5-28
usage, B-2
Modbus slaves
station address, 3-12, 14-23
Mode transition
stop-to-run, 5-11
Modem, Hayes-compatible, 14-16
Modes of operation, CPU, 5-10
Module hardware failure, 15-25
Module interrupts, 6-19
Modulus, 8-120, 9-25
Move Data, 8-92, 9-18
Multiple I/O scan sets, 5-22
Multiply, 8-121, 9-26

N

Name Server IP address, 4-3
NaN (Not a Number)
defined, 7-21
PACSystems vs. others, C-13
Negate, 9-27
Nested calls, 6-2
New features, 1-2
No Fault Contact, 8-33
No user program on power-up, 15-32
Noncritical CPU software event, 15-37
Normal block scheduling, 6-19
Normal sweep mode
application program task execution, 5-4
programmer communications window, 5-4
system communications window, 5-5
Normally closed and normally open contacts, 8-34
Not Equal, 8-137, 9-12
Null system configuration for RUN mode, 15-34
Numerical data, 7-19

O

OEM protection, 5-20
Off Delay Timer, 8-144
On Delay Stopwatch Timer, 8-146
On Delay Timer, 8-149
One-shot coil, 8-28
Online editing, 6-4, 15-31, C-13
Online testing, C-13
Operands for instructions, 7-24
Operation, Protection, and Module Status, 2-1
Operators, Structured Text, 12-2
Option module
dual port interface tests, 5-27
self-test completion, 5-27
Option module software failure, 15-25
Output references (%Q), 7-8
Output scan, 5-4
Overflow
floating point numbers, 7-21
math functions, 8-115, 9-22
math functions, 9-22
Overhead sweep impact times, A-12
base sweep time, A-12
calculating predicted sweep times, A-27
I/O interrupt performance and sweep impact, A-25
I/O module sweep impact times worksheet, A-18
I/O scan and I/O fault sweep impact, A-15
intelligent option modules, A-24
programmer sweep impact time, A-14
sweep impact of Genius I/O and GBCs, A-19
sweep impact of I/O modules, A-15
Overrides, 7-11

P

PACSystems
vs. Series 90, C-2
Parameter passing mechanisms, 6-13

Parameterized block, 6-4
and local data, 6-4
reference out of range, 6-4
referencing formal parameters, 6-4
versus PSB, C-8
Part numbers
station manager cable, 2-6
Password access failure, 15-34
Passwords, 5-19
enabling after disabled, 5-20
PCM, 13-9
C code conversion, C-24
Permanent datagrams, 14-52
PID function
control block, 11-4
reference array, 11-4
time interval, 11-11
Pin assignments
Embedded Ethernet port, 13-2
serial ports, 13-4
PING Station Manager command, 4-7
Pinging the TCP/IP Interfaces on the Network, 4-7
PLC system fault table full, 15-29
Point faults, 15-13
Port Status, read, 14-12
Power-down sequence, 5-28
Power-up self-test, 5-26
Power-up sequence, 5-26
CPU memory validation, 5-26
I/O system initialization, 5-27
logic/configuration source, 5-14
option module dual port interface tests, 5-27
option module self-test completion, 5-27
power-up self-test, 5-26
system configuration, 5-27
Preemptive block scheduling, 6-20
Privilege levels, 5-20
Program block
how blocks are called, 6-2
program blocks and local data, 6-12
Program block checksum failure, 15-27
Program execution
controlling, 6-16
row-major vs. column-major, 6-14, C-3
Program Flow functions, 8-125, 9-29
Program name, 6-1, C-2
Program register references (%P), 7-6
Program scan, 5-4
Program structure
how blocks are called, 6-2
program blocks and local data, 6-12
Programmable Coprocessor Module, 13-9
Programmer sweep impact times, A-14
Protection level request, 5-20
Protocol errors, 14-7
Protocols supported, 13-3
PSB (parameterized subroutine block), C-8

R

Range function, 8-138
Read Bytes, 14-19
Read controller ID, 10-21
Read controller run state, 10-22
Read elapsed power down time, 10-44
Read elapsed time clock, 10-27, 10-48
Read fault tables, 10-33
Read IO forced status, 10-31
Read last-logged fault table entry, 10-25
Read master checksum, 10-41
Read or change TOD clock, 10-12
Read String, 14-21
Read sweep time, 10-19, 10-50
Read switch postion, 8-52
Read target name, 10-20
Read window modes and times, 10-6
REAL numbers
internal format of, 7-21
Redundant IP, 4-4
References, 7-6
associated transitions and overrides, 7-11
data scope, 7-13
discrete references, 7-8
fault locating, 7-12, 15-11
indirect, 7-6
register references, 7-6
size and default value, 7-9
system fault references, 15-8
system status (%S), 7-14
Related documents, 1-8
Relational functions, 8-135, 9-10
Reset module, 10-42
Reset of IOC, 15-56
Reset of, addition of, or extra option module, 15-17
Reset watchdog timer, 10-19
Retentiveness
of data memory across power failure, 5-28
of logic and data, 7-12
variables associated with coils, 8-25
Rotate Bits, 8-22
Row-major logic execution, 6-14, C-3
RTU messages, 14-32
RTU slave, 14-7
protocol, 14-23
message format, 14-23
turnaround time, 14-24
Run/stop operations, 5-10
run/outputs disabled, 5-10
run/outputs enabled, 5-10

serial protocol configuration, 3-11
stop mode protocol configuration, 13-4
stop/IO scan, 5-10
stop/No IO scan, 5-10
switch enable/disable, 3-2

S

Scale, 8-123
Scan parameters, 3-4
Scan sets
analog base/expansion modules, C-21
operation, 5-22
parameters, 3-15
Scope
data, 7-13
Security, system, 5-19
privilege levels, 5-20
Self-test
I/O system initialization, 5-27
option module dual port interface tests, 5-27
option module self-test completion, 5-27
power-up self-test, 5-26
Serial I/O
Cancel Operation function, 14-15
Flush Input Buffer function, 14-11
Initialize Port function, 14-10
Input Buffer function, 14-11
Read Bytes function, 14-19
Read Port Status function, 14-12
Read String function, 14-21
Write Bytes function, 14-16, 14-18
Write Port Control function, 14-14
Serial ports
CPE010, CPE020, CRE020, 2-2
CPE030/CRE030, CPE040/CRE040, 2-5
CPU parameters, 3-11
CPU310, 2-12
CPU320, 2-14
station manager parameters, 4-5
Service requests, 10-1
example, 10-2
Set run enable/disable, 10-32
Set, Reset Coil, 8-26
Setting loop gains for PID
Ideal tuning, 11-18
Ziegler and Nichols tuning, 11-17
Settings
CPU, 3-2
Shift Bits, 8-23
Shut down CPU, 10-23
Skip next I/O scan, 10-47
Slot location, RX3i CPU, C-29
SNP master, 14-7
SNP slave protocol, 14-52
Specifications
CPE010, CPE020, CRE020, 2-3
CPE030/CRE030, CPE040/CRE040, 2-6
CPU310, 2-13
CPU320, 2-15
Square Root, 8-4
SRTP communication with older clients
RX3i, C-28
Stack
increasing for converted folders, C-27
Station address
Modbus slaves, 3-12, 14-23
Status address location, 4-4
STOP mode, 5-10
Storing configuration, 3-17
Structure of application programs, 6-1
Structure variables, 6-7
Structured Text
expressions, 12-1
language, 6-16, 12-1
operators, 12-2
statement types, 12-4
syntax, 12-3
Structured Text statement types
argument present, 12-18
assignment, 12-5
CASE, 12-12
EXIT, 12-19
FOR, 12-14
function call, 12-6
IF, 12-11
REPEAT, 12-17
RETURN, 12-10
WHILE, 12-16
Subnet mask, 4-3
Subroutines
Call function, 6-16
Subtract, 8-124, 9-28
Suspend I/O, 8-55
Suspend/resume IO interrupt, 10-45
Swap function, 8-96
Sweep impact, A-12
Ethernet global data, A-21
GBC, A-19
Genius I/O, A-19
I/O scan and I/O fault, A-15
intelligent option, A-24
local I/O, A-15
programmer, A-14
timed interrupt, A-26
Sweep, CPU, 5-2
base sweep time, A-12
modes, 5-6
STOP mode, 5-10

Switches
CPU reset, 2-1
Ethernet restart, 2-11
read switch position function, 8-52
Run/Stop mode, 2-1
Symbolic variables, 7-2
System bus error, 15-23
System configuration, 5-27
System configuration mismatch, 15-18
System fault references, 15-8
System operation
clocks and timers, 5-16
I/O system, 5-21
passwords, 5-19
power-down sequence, 5-28
power-up sequence, 5-26
retention of data memory across power failure, 5-28
system security, 5-19
System register references (%R), 7-6
System status references (%S), 7-8, 7-14
%S0020, C-2

T

Technical support, 1-9
Temporary references (%T), 7-8
Time tick references, 7-14
Timed contacts, 7-14, 8-139, C-12
Timed interrupts, 6-18
performance impact, A-26
Time-of-day clock, 5-16
accuracy, 2-3, 2-6, 2-13, 2-15
reading and setting, 5-16, 10-12
synchronizing to SNTP server, 5-17
Timers, 5-16, 8-139, 8-140, 9-30
conversion from Series 90-30, C-12
function block data, 8-71
in function blocks, 8-143
in parameterized blocks, 8-141
watchdog timer, 5-18
Timing, instruction, A-2
Transfer List parameters, 3-10
Transition Coils, 8-28
comparison, 8-30
POSCOIL and NEGCOIL, 8-28
PTCOIL and NTCOIL, 8-29
Transition contacts, 8-35
comparison, 8-38
POSCON and NEGCON (legacy), 8-35
PTCON and NTCON (IEC), 8-37
Transitions, 7-11
Trig Functions, 8-5
Truncate, 8-70
Turnaround time
RTU slave, 14-24

U

UDFBs
defining, 6-6
instance data, 6-7
instances, 6-7
internal variables, 6-9
logic restrictions, 6-9
parameters, 6-8
scope, 6-7
Up Counter, 8-73
User references, 7-6
system fault references, 15-8
User-defined faults, 15-5
logging, 10-38

V

Variables, 7-2
C, initialization, 6-11
I/O, 7-3
mapped, 7-2
member, 6-6
symbolic, 7-2
VME addressing
PACSystems vs. Series 90-70, C-23
VME_ functions, 8-78, C-23, C-26

W

Watchdog timer, 5-18
restarting, 5-18
Window completion failure, 15-34
Window modes, 5-9
Constant Window mode, 5-9
Limited mode, 5-9
Run-to-Completion, 5-9
Wires, 8-134
Word references, 7-6
Word register references (%W), 7-6
Word-for-word changes
attempting to correct parameterized block reference, 6-4
defined, 7-23
privilege level, 5-19
symbolic variables, 7-23
Write Bytes, 14-18

Y

Y0 parameter, 6-4

Z

Ziegler and Nichols tuning, 11-17