GFK-0265

New In Stock!

GE Fanuc Manuals

programming-software

1-919-535-3180

Series 90-70 PLC CPU Instruction Set Reference Manual

GFK-0265

New In Stock!

~~GE Fanuc Manuals~~

programming-software
1-919-535-3180

Series 90-70 PLC CPU Instruction Set Reference Manual

GE Fanuc Automation

Programmable Control Products

Series 90™-70 PLC
CPU Instruction Set

Reference Manual

Warnings, Cautions, and Notes as Used in this Publication

<div style="text-align:center">

Warning

</div>

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.

<div style="text-align:center">

Caution

</div>

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

<div style="text-align:center">

Note

</div>

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 Automation assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Fanuc Automation 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 Automation North America, Inc.

Alarm Master	Genius	ProLoop	Series Three
CIMPLICITY	Helpmate	PROMACRO	VersaMax
CIMPLICITY 90-ADS	Logicmaster	Series Five	VersaPro
CIMSTAR	Modelmaster	Series 90	Vumaster
Field Control	Motion Mate	Series One	Workmaster
GEnet	PowerTRAC	Series Six	

This manual describes the system operation, fault handling, and Logicmaster 90-70 programming instructions for the Series 90™-70 programmable controller. The Series 90-70 PLC is a member of the Series 90™ family of programmable logic controllers from GE Fanuc Automation.

Revisions to This Manual

The following changes have been made to this manual to reflect feature changes, corrections, and updates to existing information:

*   References made to CPX and CGR model CPUs, where appropriate, throughout the
    ŷ ` ` ` ` □□
*   Value for Constant Sweep timer corrected (chapter 2, pg. 2-46).
*   Note added after Table 2-18 regarding CPU Mode switch and description of privilege level 1 updated in table. (chapter 2, page 2-79)
*   Description of System Faults updated (chapter 3, pg. 3-2)
*   Chapters 4 through 12 contain information that was presented in a single chapter (Chapter 4) in previous versions. This information has been divided into separate chapters to improve access to the programming instruction descriptions.
*   Appendix A, CPU Performance Data, tables revised (all information not available, will be added to a future version)
*   Paragraph added , beginning with "Each Ethernet Global . . . . ", page A-24
*   Section titled "Relative CPU Test Performance" added at end of Appendix A
*   Other corrections and clarifications as necessary

Content of This Manual

Chapter 1. Introduction: provides an overview of the Series 90-70 PLC system and the Series 90-70 instruction set.

Chapter 2. System Operation: describes certain system operations of the Series 90-70 PLC system. This includes a discussion of the PLC system sweep sequences, the system power-up and power-down sequences, clocks and timers, security, I/O, and fault handling. It also includes general information for a basic understanding of programming ladder logic.

Chapter 3. Fault Explanations and Correction: provides troubleshooting information for a Series 90-70 PLC system. It explains fault descriptions in the PLC fault table and fault categories in the I/O fault table.

Chapters 4 — 12. Series 90-70 Instruction Set: describes programming instructions available for Series 90-70 PLCs. These chapters correspond to the main program function groups.

Appendix A. CPU Performance Data: lists the memory size in bytes and the execution time in microseconds for each programming instruction. Memory size is the number of bytes required by the function in a ladder diagram application program. Appendix A also contains timing information for other PLC tasks which, when used in conjunction with the instruction timings, can be used to predict CPU sweep times. Refer to Appendix F for IEEE format when dealing with floating-point math operations.

Appendix B. Interpreting Fault Tables: describes how to interpret the message structure format when reading the fault tables using Logicmaster 90-70 software.

Appendix C. Instruction Mnemonics: lists mnemonics that can be typed to display programming instructions while searching or editing a program. Provides a worksheet for use in determining the total number of bytes of user data used and how much is still available for the user program.

Appendix D. Memory Allocation: provides a worksheet for determining the total number of bytes of user data used and how much is still available for the user program.

Appendix E. Key Functions: lists the special keyboard assignments used for the Logicmaster 90 software.

Appendix F. Using Floating-Point Numbers: describes special considerations for using floating-point math operations.

Related Publications

Logicmaster™ 9070 Programming Software User's Manual   (GFK-0263).

Logicmaster™ 9070 Important Product Information   (GFK-0350).

Series 90™70 Programmable Controller Installation Manual   (GFK-0262).

Series 90™ Programmable Coprocessor Module and Support Software User's Manual
(GFK-0255) .

Series 90™ PCM Development Software (PCOP) User's Manual (GFK-0487).

C Programmer's Toolkit for Series 90™70 PLCs User's Manual   (GFK-0646).

Series 90™ Sequential Function Chart Programming Language User's Manual (GFK-0854).

MegaBasic™ Programming Language Reference Manual (GFK-0256).

CIMPLICITY™ 90ADS Alphanumeric Display System User's Manual   (GFK-0499).

CIMPLICITY™ 90ADS Alphanumeric Display System Reference Manual   (GFK-0641).

Alphanumeric Display Coprocessor Module Data Sheet (GFK-0521).

Series 90™70 Genius I/O System User's Manual   (GEK-90486-1).

Series 90™70 Genius I/O Analog and Discrete Blocks User's Manual   (GEK-90486-2).

Workmaster® II PLC Programming Unit Guide to Operation (GFK-0401).

Series 90™70 Genius Bus Controller User's Manual   (GFK-0398).

Series 90-70 FIP Bus Controller User's Manual (GFK-1038).

Guidelines for the Selection of Third-Party VME Modules   (GFK-0448).

Series 90™ Ethernet Communications User's Manual (GFK-0868).

Series 90™ MAP 3.0 Communications User's Manual (GFK-0869).

TCP/IP Ethernet Communications for the Series 90 PLC User's Manual (GFK-1541)

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Introduction

The Series 90™-70 PLC is a member of the GE Fanuc Series 90™ PLC family of programmable logic controllers (PLCs). It is easy to install and configure, offers advanced programming features, and is designed for compatibility with other PLCs offered in the Series 90 family of PLCs. Through the use of the latest design and manufacturing technology, open architecture VME bus, and the ability to connect to Genius and FIP I/O, the Series 90-70 PLC provides a powerful, cost-effective platform for small applications through the very largest. This manual discusses the features of the Release 7.80, and later Series 90-70 PLC.

Software Architecture

The programming software architecture provides a platform upon which to build structured control programs. Programs may be built from many program blocks, each of which is related to a control function. Structured programs permit parallel development of a complete program as a collection of program blocks developed independently by many different individuals or OEMs. Structured programs are also easier to understand and debug. A control program may be built of many smaller program blocks, each of which can relate to a specific machine function. This approach makes it easier to isolate and associate control logic with machine functions.

Beginning with Release 6 PLC CPUs, it has been possible to incorporate multiple programs into a folder. All of these programs can be written in C or one program can be an RLD (Relay Ladder Diagram language) or SFC (Sequential Function Chart language) program with the remaining programs written in C. In addition, Release 6 and later PLCs have built-in debugging capabilities for C programs and external blocks. For more information on this feature, refer to the *C Programmer's Toolkit* manual (GFK-0646).

Note that Logicmaster 90-70 does not support many of the features new to release 7.0 and later CPUs, such as Ethernet Global Data, I/O Scan Sets, VME 3rd party Interrupts, and Bulk Memory Access (BMA).

Terminology Used in This Manual

The following terms are used with their defined meanings throughout this manual:

User program: any user-generated code, that is, an RLD program, an SFC program, or a standalone C program

Block: any RLD block, Parameterized Subroutine Block, or external block; an external block being either a C block or a C function block (CFBK)

Fault Handling

Faults are handled by a software alarm processor function that time-stamps and logs system and I/O faults in two tables (the PLC fault table and the I/O fault table). These tables can be displayed on the programming software screen or uploaded to a host computer or other coprocessor. Application programs can also gain access to the fault information.

Hardware Configuration

Configuration is the process of assigning logical addresses, as well as other characteristics, to the hardware modules in the system. It may be done either before or after programming; however, it is recommended that configuration be done first.

Using This Manual

This manual is distributed with Logicmaster 90 programming software, and describes the PLC hardware and programming features available in the CPU. Refer to the IPI distributed with Logicmaster 90 for CPU and programming features not described in this version of the manual.

Reference information is available in this manual, as described below:

Appendix A lists the memory size in bytes and the execution time in microseconds for each of the programming instructions. Appendix A also contains timing information for other PLC tasks which, when used in conjunction with the instruction timings, can be used to predict CPU sweep times.

Appendix B describes how to interpret the message structure format when reading the PLC and I/O fault tables.

Use the worksheet in Appendix C to determine the total number of bytes of user data used and how much is still available for the user program.

Refer to Appendix D for IEEE format when dealing with floating-point math operations.

Chapter	
2	*System Operation*

This chapter describes certain system operations of the Series 90-70 PLC system. The table displayed below summarizes the content of each section in this chapter.

Section	Title	Description	Page
1	Basic PLC Sweep Summary	Describes the major steps in a typical PLC sweep, including application program task execution, Programmer Communications Window, System Communications Window, and Background Window.	2-2
2	User Reference Data	Describes user reference data, system status/fault references, and data types.	2-11
3	Program Organization	Describes the structure and use of LD blocks, PSB blocks, external blocks, and standalone C programs.	2-27
4	PLC Sweep Modes and Program Scheduling Modes	Explains Normal Sweep, Constant Sweep, Constant Window, Microcycle Sweep, and Stop modes. Also describes Triggered Interrupt blocks/programs and timed interrupts.	2-48
5	Run/Stop Operations	Describes the four modes of operation supported by the 90-70 PLC: Run/Outputs Enabled, Run/Outputs Disabled, Stop/IO Scan, and Stop/No IO Scan.	2-70
6	Power-Up and Power-Down Sequences	Describes the three parts of system power-up (including power-up self-test, PLC operation initialization, and system configuration), the power-down sequence, and the retention of data memory.	2-72
7	Clocks and Timers	Describes the elapsed time clock, time-of-day clock, and watchdog timers.	2-76
8	System Security	Describes protection level request from the programmer, including password assignment and block lock, OEM protection and password, and the write protect keyswitch.	2-78
9	Series 90-70 PLC I/O System	Describes I/O data mapping and diagnostic data.	2-81

Section 1: Basic PLC Sweep Summary

The user program(s) in the Series 90-70 PLC execute in a repetitive fashion until stopped by a command from the programmer or a command from another device or from the Run/Stop toggle switch on the CPU module. In addition to executing the user program(s), the PLC obtains data from input devices, sends data to output devices, performs internal housekeeping, services the programmer, services other communications, and performs self-tests. The sequence of operations necessary to execute these components one time is called a sweep. This section summarizes the sweep phases; for more detailed information, refer to section 4 of this chapter.

Basic PLC Sweep

There are seven major phases in a typical PLC sweep as shown in the following figure:
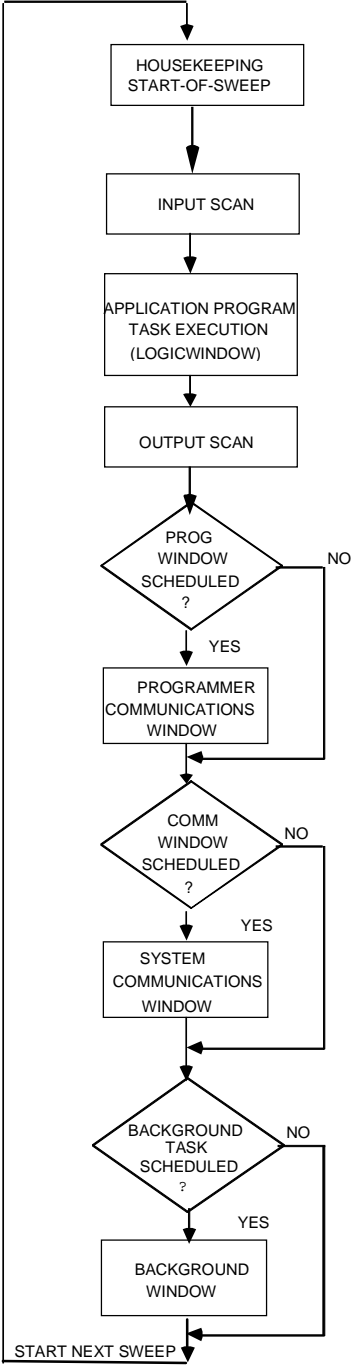
Figure 2-1. Phases of a Typical PLC Sweep

Table 2-1. Major Phases in a Typical PLC Sweep

Step	Description
Housekeeping	Updating %S bits, determining timer update values, and determining the sweep mode occur in this phase.
Input Scan	The CPU reads input data from Bus controllers and input modules during this phase.
Application Program Task Execution (Logic Window)	The CPU solves the logic program(s), using data obtained from the input devices and sets bits to affect the state of output devices.
Output Scan	The CPU writes output data to Bus controllers and output modules during this phase. The user program checksum is computed during this phase of the sweep except when in Microcycle sweep mode.** Polling for faulted boards also occurs during this phase.
Programmer Communications Window	Communication with the programmer when using serial and WSI devices occurs here with data and/or status transfer in both directions. In addition, reconfiguration of a module or rack also occurs during this portion of the sweep.
System Communications Window	Communications with all intelligent devices (except the Serial or WSI programmer when using a serial or WSI connection) occur during this window. For example, supplying data to a PCM* that is driving a process display would occur during this window. The Ethernet programmer communicates in the System Communications Window.
Background Task Window	CPU self-tests occur in this window.

*For information about the PCM, refer to the *Series 90™ Programmable Coprocessor Module and Software Support* (GFK-0255).
**In Microcycle sweep mode the user program checksum is computed during the Input Scan.

Housekeeping

The housekeeping portion of the sweep performs all of the tasks necessary to prepare for the start of the sweep. This includes updating %S bits, determining timer update values, and determining the mode of the sweep (Stop or Run).

Input Scan

The scanning of the inputs occurs just prior to the logic solution. During the input scan, the CPU reads input data from the Genius Bus Controllers, FIP Bus Controllers, and Series 90-70 input modules. Also, the Ethernet Global Data for Consumed exchanges is read from the Ethernet module into PLC memory. For details, see the *TCP/IP Ethernet Communications for the Series 90 PLC User's Manual,* GFK-1541.

When referring to FIP in this scan, only periodic VCOM (MPS) services are affected. Messages are received in the System Communication Window.

Series 90-70 I/O modules are scanned from lowest to highest I/O reference address. There is no guaranteed scanning order for Bus Controllers.

Note

The input scan will not be performed if a program has an active Suspend I/O function on the previous sweep.

Application Program Task Execution (Logic Window)

The Logic Window is the phase of the sweep where user programs execute. Immediately following the completion of the input scan, the PLC Executive determines which user program(s) are to be run. Programs are then resumed and/or invoked as necessary. Solving the logic provides a new set of outputs.

Interrupt programs and blocks can execute during any phase of the sweep. Refer to section 4 for further details.

There are many ways in which program execution can be controlled to meet the system's timing requirements. The following is a partial list of the commonly used methods:

- JUMP functions can be used to skip portions of the logic.
- 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 and programs are called more or less frequently, depending on their importance and on timing constraints.
- Microcycle sweep mode can be used to phase programs which need to run less often while limiting the logic window execution time.

A list of execution times for instructions can be found in Appendix A.

Note

In Microcycle Sweep mode, the Logic Window can be skipped or preempted as necessary by the PLC Executive.

Output Scan

Outputs are scanned immediately following logic solution. During the output scan, the CPU sends output data to the Genius Bus Controllers, FIP Bus Controllers, and Series 90-70 output modules. Also, the PLC, as the producer in an Ethernet Global Data exchange will periodically produce new samples of data for use by the configured devices on the Ethernet network. (For details, see GFK-1541, the TCP/IP Ethernet Communications for the Series 90 PLC User's Manual).

Series 90-70 output modules are scanned from lowest to highest I/O reference address. Bus Controllers are scanned from rack 0 to rack 7 and lowest to highest slot number within each rack.

Note

The output scan will not be performed if a program has an active Suspend I/O function on the current sweep.

When referring to FIP in this scan, only periodic VCOM (MPS) services are affected. Messages are received in the System Communication Window.

Polling for faulted boards also occurs during the output scan phase of the sweep. Faulted board polling recognizes replacement boards for faulted ones and reconfigures them into the system. If a board that was previously in the system or configured by the user to be in the system is listed as

faulted, it must be polled periodically to determine if a new board has replaced it. Once a previously faulted board is detected as no longer faulted, reconfiguration is run in the Programmer Communications Window until the board(s) are reconfigured into the system.

The background checksum calculation also occurs during the output phase of the sweep. During each output scan phase of the sweep, the configured amount of words of user program are included in the checksum calculation. This checksum helps to ensure the integrity of the user logic while the CPU is executing. If the CPU is configured to perform a background checksum calculation (16 is the default), then this part of the output phase is performed; otherwise, it is skipped.

There are other tests performed during the Output Scan: Processor test–tests basic operation of the microprocessor and BCP Opcode test–tests basic operation of all BCP instructions.

Note

Beginning with the Release 7 CPUs, for Microcycle Sweep only, the background checksum calculation will occur during the input phase of the sweep.

Programmer Communications Window

This part of the sweep is dedicated to serial and parallel communications with the programmer and performing faulted board reconfiguration. This is also when communication with the C debugger occurs. If there is a programmer attached, a debugging session is active, or if there is a board in the system that requires reconfiguration (as detected during the faulted board polling portion of the sweep), the CPU executes the Programmer Window. The Programmer Window will not execute if there is no programmer attached, no active debug session occurring, and no board to be configured in the system. During the Programmer Window, highest priority is given to board configuration. Boards are configured as needed, up to the total time allocated to the Programmer Window.

The built-in SNP connection and the parallel programmer connection (the two dedicated programmer ports on most systems) communicate to the CPU through the Programmer Window. For the CPX models, there are three built-in SNP ports. All three of these communicate through the Programmer Window. The CPU will complete any previously unfinished requests and then begin to process any pending requests in the queue. When the time allocated for the window expires, processing stops.

The Programmer Window time defaults to 10 milliseconds. This value can be configured and stored to the PLC or it can be changed online using your programming software.

Time and execution of the Programmer Window can also be dynamically controlled from the user program using Service Request function #3. The Programmer Communications Window time can be set to a value from 0 to 255 milliseconds. Note that if the Programmer Communications Window is set to 0, there are 3 ways to again open the window; perform a batteryless power-cycle, go to STOP mode, or use the parallel programming port as noted below.

Note

Even if the Programmer Window is skipped, the PLC can still respond to commands to change mode or state, or to redefine the Programmer Window if the programmer is attached through the parallel port on the Bus Transmitter Module (BTM), or by manually putting the PLC into STOP mode.

System Communications Window

The System Communications Window is the part of the sweep used for communication between the CPU and intelligent modules such as the PCM, Genius Bus Controller, FIP Bus Controller, and TCP/IP Ethernet modules. Note that the Ethernet programmer communicates in the System Communications Window.

At the start of the System Communications Window, the CPU will complete any previously unfinished request before executing any pending requests in the queue. When the time allocated for the window expires, processing stops.

The System Communications Window defaults to "Run to Completion" mode. This means that all currently pending requests on all intelligent option modules are processed every sweep. A different mode can be configured and stored to the PLC, or it can be changed online using your programming software.

Time and execution of the System Communications Window can also be dynamically controlled from the user program using Service Request function #4. This allows communications functions to be skipped during certain time-critical sweeps. The System Communications Window time can be set to a value from 0 to 255 milliseconds.

Background Window

A CPU self-test is performed in this window. Included in this self-test is a verification of the checksum for the 90-70 CPU operating system software.

The Background Window time defaults to 0 milliseconds. A different value can be configured and stored to the PLC, or it can be changed online using your 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.

Window Modes

The previous sections have described the phases of a typical PLC sweep. The Programmer Window, System Communications Window, and Background Window phases of the PLC sweep can be run in various modes, based on the PLC Sweep mode. (PLC sweep modes are described in detail in section 4.) 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 PLC will transition to the next phase of the sweep.

Constant   In Constant Window mode, the total amount of time that the Programmer Communications Window, System 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 PLC will cycle through these windows the specified amount of time polling for further requests. If any window is put in constant window mode, all will be 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 PLC will proceed to the next phase of the sweep.

Data Coherency in Communications Windows

When running in Constant or Limited Window mode, the Programmer and System Communications Windows may be terminated early in all PLC sweep modes. If an external device, such as a GBC (Genius Bus Controller), 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 PLC 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 PLC 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 PLC RUN/STOP state. Process graphics will often indicate everything is still operating normally.

Also, note that non-retentive outputs do not clear until PLC CPU is changed from STOP to RUN.

CPU Sweep in STOP Mode

The 90-70 PLC has two modes of operation while it is in Stop mode: Stop/NoIO and Stop/IOScan.

When the PLC is in Stop/NoIO mode the Input Scan, Logic Window, and Output Scan phases of the PLC sweep are skipped.

When the PLC is in Stop/IOScan mode the Logic Window phase of the PLC is skipped but the Input Scan and Output Scan phases are performed each sweep.

In both Stop/NoIO and Stop/IOScan modes, the two Communications Windows run in Run-to-Completion mode and the Background Window runs in Limited mode with a 10 millisecond limit.

```
              ┌──────────────────────┐
              │   START-OF-SWEEP      │
              │   HOUSEKEEPING        │
              └──────────────────────┘
                         │
              ┌──────────────────────┐      EXECUTES
              │   INPUT SCAN         │◁     IN STOP/IOSCAN
              └──────────────────────┘      ONLY
                         │
              ┌──────────────────────┐      EXECUTES
              │   OUTPUT SCAN        │◁     IN STOP/IOSCAN
              └──────────────────────┘      ONLY
                         │
              ┌──────────────────────┐      RUNS
              │   PROGRAMMER         │◁     TO
              │   COMMUNICATIONS     │      COMPLETION
              │   WINDOW             │
              └──────────────────────┘
                         │
              ┌──────────────────────┐      RUNS
              │   SYSTEM             │◁     TO
              │   COMMUNICATIONS     │      COMPLETION
              │   WINDOW             │
              └──────────────────────┘
                         │
              ┌──────────────────────┐      LIMITED
              │   BACKGROUND         │◁     (10 MS)
              │   WINDOW             │
              └──────────────────────┘
```
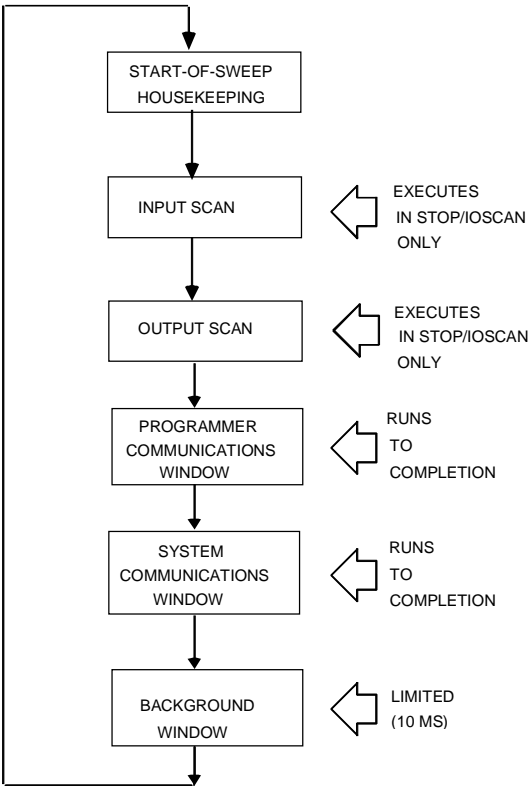
Figure 2-2. CPU Sweep in Stop/NoIO and Stop/IOScan Mode

Note

Stop/IOScan is not supported in Microcycle Sweep mode.

PLC Sweep Modes

The 90-70 PLC supports four PLC sweep modes:

Normal Sweep   In Normal Sweep mode, each PLC 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 a Limited or Run-to-Completion mode.

Constant Sweep   In Constant Sweep mode, each PLC sweep begins at a user-specified Constant Sweep time after the previous PLC sweep began. The Logic Window is executed in its entirety each sweep. If there is sufficient time at the end of the sweep, the PLC will alternate among the Communications and Background Windows, allowing them to execute until it is time for the next sweep to begin.

Constant Window   In Constant Window mode, each PLC sweep can consume a variable amount of time. The Logic Window is executed in its entirety each sweep. The PLC will alternate among the Communications and Background Windows, allowing them to execute for a time equal to the user-specified Constant Window timer.

Microcycle Sweep   In Microcycle Sweep mode, like Constant Sweep mode, each PLC sweep takes a fixed amount of time. The total sweep time (base cycle time) and the total time for the Communications and Background Windows is specified by the user. The Logic Window can be preempted in order to maintain the total sweep time and the Communications Windows and Background Window times. To satisfy the specified window times, the PLC alternates among the Programmer Communications Window, the System Communications Window, and the Background Window, allowing them to execute until it is time for the next sweep to begin.

Note

The information presented above summarizes the different sweep modes. For detailed information on PLC Sweep Modes, refer to "PLC Sweep Modes and Program Scheduling Modes" in section 4 of this chapter.

Section 2: User Reference Data

User References

The PLC data used in an application program is stored as either discrete or register references.

Table 2-2. Register References

Type	Description
%R	Use the prefix %R to assign system register references which will store program data such as the results of calculations.
%AI	The prefix %AI represents an analog input register. This prefix is followed by the register address of the reference (for example, %AI0015). 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. This prefix is followed by the register address of the reference (for example, %AQ0056). An analog output register holds the value of one analog output or other non-discrete value.
%P*	Use the prefix %P to assign program register references which will store program data from 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. (For more information, refer to the Appendix D, "Memory Allocation.") The %P references are not normally accessible from external hosts.
%L*	Use the prefix %L to assign local register references which will store program data unique to a block. The size of the %L data block is based on the highest %L reference in the associated block. (For more information, refer to Appendix D, "Memory Allocation.") The %L references are accessible only from within the local block.

* These reference types are scoped at a program level and are therefore only visible to LD programs.

Note

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

Indirect References

You can use indirect referencing for all register references (%R, %AI, %AQ, %P, and %L) to identify a location in memory that contains the offset in the same memory type of the data to be used. Indirect references are entered in the same way as direct references, except that the @ character is used in place of the *%* character. For example, if %R00101 contains the value 1000, then @R00101 would instruct the PLC to use the data location of %R01000.

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

Table 2-3. Discrete References

Type	Description	Retentiveness
%I	The %I prefix represents input references. This prefix is followed by the reference's address in the input table (for example, %I00121). %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.	Always retentive
%Q	The %Q prefix represents physical output references. The coil check function checks for multiple uses of %Q references with relay coils or outputs on functions. Beginning with Release 4 of the Logicmaster, you can select the level of coil checking desired (Single, Warn Multiple, or Multiple). The %Q prefix is followed by the reference's address in the output table (for example, %Q00016). %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. *	Based on type of coil used
%M	The %M prefix 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. *	Based on type of coil used
%T	The %T prefix 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 never retained through power loss or Run-to-Stop-to-Run transitions and cannot be used with retentive coils.	Always non-retentive

Table 2-3. Discrete References - Continued

Type	Description	Retentiveness
%S %SA %SB %SC	The %S, %SA, %SB, and %SC prefixes represent system status references. These references are used to access special PLC data such as timers, scan information, and fault information. For example, the %SC0012 bit can be used to check the status of the PLC 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.	Retentive, yet always initialized at power-up (See description for behavior of each individual bit in Table 2-9.)
%G %GA %GB %GC %GD %GE	The %G, %GA, %GB, %GC, %GD, and %GE prefixes represent global data references. These references are used to access data shared among several PLCs. %G, %GA, %GB, %GC, %GD, and %GE references can be used on contacts and retentive coils because the memory is always retentive. %G, %GA, %GB, %GC, %GD, and %GE cannot be used on non-retentive coils.	Always retentive

* Retentiveness is based on the type of coil. For more information, refer to "Retentiveness of Logic and Data" later in this section.

User Reference Size and Default

Maximum user references and default reference sizes for each model of CPU are listed in the tables below.

Table 2-4. User Reference Sizes

Item	CPU Model					
	935/928 925/915/790	924/914	788	780/781 782/789	771/772	731/732
Maximum %I reference	12288 points	12288 points	352 points[1]	12288 points[2]	2048 points[2]	512 points[2]
Maximum %Q reference	12288 points	12288 points	352 points[1]	12288 points[2]	2048 points[2]	512 points[2]
Maximum %M reference	12288 points	12288 points	12288 points	12288 points	4096 points	2048 points
Maximum %T reference	256 points	256 points	256 points	256 points	256 points	256 points
%S total (S, SA, SB, SC)	512 points	512 points	512 points	512 points	512 points	512 points
%G (GA, GB, GC, GD, GE)	7680 points	7680 points	7680 points	7680 points	7680 points	1280 points
User RAM	1024K bytes (6MB for 928)	512K bytes	512K bytes	512K bytes (CPX 782 has 1024 KB)	CPU771/2: 64, 128, 256, 512 KB depending on expansion memory board purchased CPX772: 512 KB	32K bytes
Maximum %AI reference	8K words	8K words	8K words	8K words	8K words	8K words
Maximum %AQ reference	8K words	8K words	8K words	8K words	8K words	8K words
Maximum %R, 1K word increments	16K words	16K words	16K words	16K words	16K words	16K words
Maximum %L (per block)	8K words.	8K words	8K words	8K words	8K words	8K words
Maximum %P	8K words	8K words	8K words	8K words	8K words	8K words

[1] Total number of physical input and output points together cannot exceed 352 points. This corresponds to approximately 100 redundant points. Refer to Chapter 1 of GFK-1277 for more information.

[2] Prior to Release 6 of Logicmaster, the programming software restricted the total %I and %Q to the limit shown individually for each. For example, when using previous programming packages with a 782 CPU, there was a maximum of 12288 points of %I and %Q *combined*. This restriction no longer exists with the newer versions of Logicmaster.

Table 2-5. Default Memory Sizes

Memory Type	CPU Model					
	935/928 925/915/790	924/914	780/781/782 788/789	781/782	771/772	731/732
%AI	64 words	64 words	64 words	64 words	64 words	64 words
%AQ	64 words	64 words	64 words	64 words	64 words	64 words
%R	1024 words	1024 words	1024 words	1024 words	1024 words	1024 words
%P	0 words	0 words	0 words	0 words	0 words	0 words
%L	0 words	0 words	0 words	0 words	0 words	0 words

%G User References and CPU Memory Locations

The Series 90-70 CPU contains only one data space for all of the global data references (%G, %GA, %GB, %GC, %GD, and %GE). The internal CPU memory for this data is 7680 bits long. Your programming software provides the user a subdivided representation by 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. Your programming software interprets the requested global reference type (%G, %GA, %GB, %GC, %GD, or %GE) and converts it to the %G memory type and correct bit offset for use by the CPU. The actual mapping is shown in the table displayed below.

Table 2-6. **%G References and Memory Locations**

Global Data Type	%G	%GA	%GB	%GC	%GD	%GE
References Used by the Programming Software	%G1–1280	%GA1–1280	%GB1–1280	%GC1–1280	%GD1–1280	%GE1–1280
Memory Locations Used by the CPU	%G1–1280	%G1281–2560	%G2561–3840	%G3841–5120	%G5121–6400	%G6401–7680

This information is useful when programming 90-70 CPU applications in C language using the C Programmer's Toolkit. For more information about using the C Programmer's Toolkit, refer to the *C Programmer's Toolkit for Series 90™ PLCs User's Manual* (GFK-0646).

Note

A 731 CPU supports **only** %G since it has only 1280 points of global data.

Genius Global Data

The Series 90-70 PLC supports the sharing of data among multiple PLC 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 PLCs that have Genius I/O capability can send and receive global data from a Series 90-70 PLC.

Transitions and Overrides

The %I, %Q, %M, and %G user references 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.

> **Caution**
>
> **Do not override transitional coils. If a transitional coil is overridden and the override is then removed, the coil will come on for one sweep. This can cause unexpected consequences in the PLC ladder logic and in field devices attached to the PLC.**

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.

Retentiveness of Logic and Data

Data is defined as retentive if it is saved by the PLC when the PLC transitions from STOP mode to RUN mode. On STOP to RUN transition, the Series 90-70 PLC preserves program logic, fault tables and diagnostics, checksums for program logic, overrides and output forces, word data (%R, %L, %P, %AI, %AQ), and bit data (%I, %G, fault locating references, and reserved bits), and %Q and %M data (unless used with non-retentive coils). %T data is non-retentive and therefore not saved on STOP to RUN transitions.

Retentive data is also preserved during battery-backed power-cycles of the PLC 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 table 2-9 for a description of the behavior of each system status reference.)

When %Q and %M references are used with non-retentive coils, they are non-retentive (that is, cleared when the PLC transitions from Stop to Run, including power-up in Run mode). Non-retentive coils include coils -()-, negated coils -(/)-, SET coils -(S)-, and RESET coils -(R)-.

When %Q or %M references are used with retentive coils or are used as function block outputs, the contents are retained through power loss and Run-to-Stop-to-Run transitions. Retentive coils include retentive coils -(M)-, negated retentive coils -(/M)-, retentive SET coils -(SM)-, and retentive RESET coils -(RM)-.

The last time a %Q or %M reference is programmed on a coil instruction determines whether the %Q or %M reference is retentive or non-retentive based on the coil type. For example, if %Q00001 was last programmed as the reference of a retentive coil, the %Q00001 data will be retentive. However, if %Q00001 was last programmed on a non-retentive coil, the %Q00001 data will be non-retentive.

Note

When only standalone C programs are used, the retentive nature of data is based solely on the memory type since there are no coil instructions. In this case %Q and %M memory types are retentive.

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.

Table 2-7. Data Scope of User Reference Data

User Reference	Range	Scope
%I, %Q, %M, %T, %S, %SA, %SB, %SC, %G, %R, %AI, %AQ, convenience references, fault locating references	System	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:
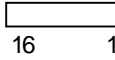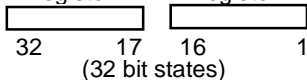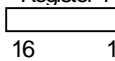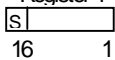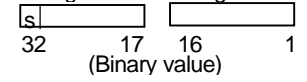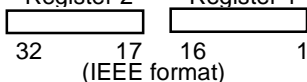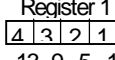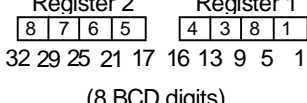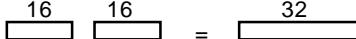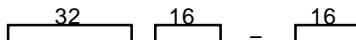
- %P should be used for program references that are shared with other blocks.

- %L are local references which 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, %AI, and %AQ references are available throughout the system.

Appendix D contains Memory Allocation formulas for determining the total number of bytes of user data used and how much is still available for the logic.

Data Types

Table 2-8. Data Types

Type	Name	Description	Data Format
BOOL	Boolean	A Boolean data type is the smallest unit of memory. It has two states, 1 or 0. A BOOL array may have length N.	
BYTE	Byte	A Byte data type has an 8-bit value. It has 256 values (0–255). A BYTE array may have length N.	
WORD	Word	A Word data type uses 16 consecutive bits of data memory. The valid range of word values is 0000 hex to FFFF hex.	Register 1 (16 bit states) 16 1
DWORD	Double Word	A Double Word data type 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.	Register 2 Register 1 32 17 16 1 (32 bit states)
UINT	Unsigned Integer	Unsigned integers use 16-bit memory data locations. They have a valid range of 0 to +65535 (FFFF hex).	Register 1 (Binary value) 16 1
INT	Signed Integer	Signed integers use 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.	Register 1 (Two's Complement value) S 16 1
DINT	Double Precision Integer	Double precision integers are stored in 32-bit data memory locations (two consecutive 16-bit memory locations) and are always signed values (bit 32 is the sign bit.) The valid range of a DINT data type is –2147483648 to +2147483867.	Register 2 Register 1 S 32 17 16 1 (Binary value)
REAL	Floating-Point	Real numbers use 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. Refer to Appendix D "Using Floating-Point Math," for IEEE format.	Register 2 Register 1 32 17 16 1 (IEEE format)
BCD-4	Four-Digit Binary Coded Decimal	Four-digit BCD numbers use 16-bit data memory locations. Each 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.	Register 1 4 3 2 1 (4 BCD digits) 13 9 5 1
BCD-8	Eight-Digit Binary Coded Decimal	Eight-digit BCD data types use 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.	Register 2 Register 1 8 7 6 5 4 3 8 1 32 29 25 21 17 16 13 9 5 1 (8 BCD digits)
MIXED	Mixed	A Mixed data type is 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.	16 16 = 32 32 16 = 16
ASCII	ASCII	Eight-bit encoded characters. A single reference is required to make up 2 (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. Command codes and non-displayable characters appear on the screen as non-alphanumeric characters (for example, @).	

S = Sign bit (0 = positive, 1 = negative).

Note

Using function blocks that are not explicitly bit-typed will affect transitions for all bits in the written byte/word/dword.

Also for information about using floating-point numbers, refer to Appendix D, "Using Floating-Point Numbers."

System Status References

System status references (formerly called "Convenience" references) in the Series 90-70 PLC are assigned to %S, %SA, %SB, and %SC memory. They each have a system-supplied nickname which enables you to enter the nickname rather than the exact %S reference. Examples of 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 3, "Fault Explanation and Correction," for more detailed fault descriptions and information on correcting faults.

While it is possible to use these special names in another context, their use is restricted (for example, you cannot use them as a block name or folder name).

Note

Most references not listed in the following table (for example, %S0002) are not used for the Series 90-70 PLC. Products that have Genius Modular Redundancy (CPU788, CPU789, and CPU790) have additional references, as does the 780 CPU and CGR935 (with CPU redundancy). Refer to GFK-1277 for status references for Genius Modular Redundancy (GMR) and GFK-1527 for CPU redundancy.

Table 2-9. System Status 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. **NOTE:** In a C stand-alone program, use a C macro to determine FST_SCN, not %S0001.
%S0003	T_10MS	0.01 second timer contact.
%S0004	T_100MS	0.1 second timer contact.
%S0005	T_SEC	1.0 second timer contact.
%S0006	T_MIN	1.0 minute timer contact.
%S0007	ALW_ON	Always ON.
%S0008	ALW_OFF	Always OFF.
%S0009	SY_FULL	Set when the PLC fault table fills up (size configurable with a default of 16 entries). Cleared when an entry is removed from the PLC fault table and when the PLC 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 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	Set to indicate a bad battery in a Release 4 or later CPU. The contact is updated when a change in the battery status occurs.
%S0121	FST_EXE	Current sweep is the first time this block has been called. Set when transitioning from Stop to Run. FST_EXE is not available to standalone C programs.
%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 PLC fault table or power cycle the CPU.
%SA0002	OV_SWP	Set when the PLC detects that the previous sweep took longer than the time specified by the user. To clear this bit, clear the PLC fault table or power cycle the CPU. Only occurs if the PLC is in Constant Sweep or Microcycle Sweep mode .
%SA0003	APL_FLT	Set when an application fault occurs. To clear this bit, clear the PLC 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 PLC 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 PLC fault table or power cycle the CPU.
%SA0011	LOW_BAT	Set when a low battery fault occurs. To clear this bit, clear the PLC fault table or power cycle the CPU.
%SA0012	LOS_RCK	Set when an expansion rack stops communicating with the PLC CPU. To clear this bit, clear the PLC fault table or power cycle the CPU.

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 PLC monitoring devices.

Also please note that if you have not fixed the condition that caused the fault, the fault may return immediately after power-cycling the PLC.

Table 2-9. System Status References - Continued

Reference	Name	Definition
%SA0013	LOS_IOC	Set when a Bus Controller stops communicating with the PLC. 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 PLC 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 PLC CPU. To clear this bit, clear the PLC 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 PLC 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 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.
%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 PLC fault table. To clear this bit, clear the PLC fault table or power cycle the CPU. For more information, see discussion of Service Request 21 in Chapter 4.
%SB0001	WIND_ER	Set when there is not enough time to start the Programmer Window in Constant Sweep or Microcycle Sweep mode, or when there is not enough time to start the Logic Window in Microcycle Sweep mode. To clear this bit, clear the PLC fault table or power cycle the CPU.
%SB0009	NO_PROG	Set when the PLC CPU powers up with memory preserved, but no user program is present. Cleared when the PLC powers up with a program present or by clearing the PLC fault table.

Table 2-9. System Status References - Continued

Reference	Name	Definition
%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 PLC fault table.
%SB0011	BAD_PWD	Set when a password access violation occurs. Cleared when the PLC fault table is cleared or when the CPU is power cycled..
%SB0012	NUL_CFG	Set when an attempt is made to put the PLC in Run mode when there is no configuration data present. To clear this bit, clear the PLC 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 PLC 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 PLC 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 PLC fault table or power cycle the CPU.
%SB0017	SBUS_FL	Set when the PLC fails to gain access to the bus. To clear this bit, clear the PLC fault table or power cycle the CPU.
%SC0009	ANY_FLT	Set when any fault occurs that causes an entry to be placed in the PLC 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 PLC fault table. Cleared when the PLC 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 PLC fault table. Cleared when the PLC 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.

Other References

The fault references are discussed in Chapter 3 of this manual but are presented here for your convenience.

Table 2-10. 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 PLC 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 PLC 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

Table 2-11. 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 PLC does not check the configuration parameters set up for individual modules such as Genius I/O blocks.

Table 2-12. Non-Configurable Faults

Non-Configurable Faults (Action)	Description
SBUS_FL (fatal)	System bus failure. The PLC CPU was not able to access the VME bus. BUSGRT-NMI error.
HRD_CPU (fatal)	PLC 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, such as the failure of a serial port on a PCM.
SFT_SIO (diagnostic)	Non-recoverable software error in a PCM or LAN interface module.
PB_SUM (fatal)	Program or block checksum failure during power-up or in Run mode.
LOW_BAT (diagnostic)	Low battery signal from CPU or another module in the system.
OV_SWP (diagnostic)	Constant sweep time exceeded.
SY_FULL, IO_FULL (diagnostic)	PLC 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 PLC 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 or Microcycle 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 Series 90 PLC supports 32 bus controllers.
STOR_ER (fatal)	Download of data to PLC from the programmer failed; some data in PLC may be corrupted.

Note

Fault and FIP locating references are discussed in Chapter 3, section 1 of this manual

Refer to the *Series 90™ Sequential Function Chart Programming Language User's Manual* (GFK-0854) for SFC references.

Section 3: Program Organization

The user program(s) contains the logic that is used to process input data and control output data. Program logic is executed repeatedly by the PLC. The Series 90-70 PLC allows up to 16 user programs, with a maximum of 1 LD program. Refer to Tables 2-4 and 2-5 for a listing of program sizes and reference limits for each CPU model.

The following figure depicts three user programs, two of which are standalone C programs. The LD program consists of four blocks (_MAIN, LD_1, PSB_X, and EXT4). The figure further illustrates the scoping of various memory types: all references except %P and %L are visible to the standalone programs; %P memory is visible to all of the program blocks, and the LD_1 block has its own local data, %L. Details of standalone programs and blocks are described later in this section.

Program "A" (Standalone)   **Program "B" (RLD)**   **Program "C" (Standalone)**

_MAIN

%P Memory

LD 1   %L

PSB X

EXT4

User Data References (%I, %Q, %R, %AI, %AQ, etc.)
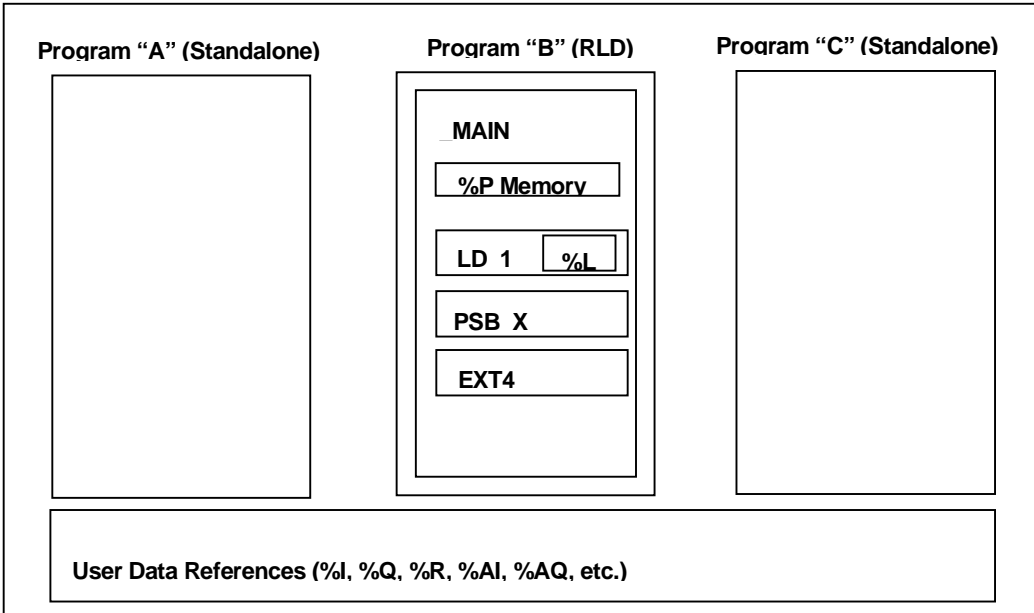
Ladder Logic Programming

An LD program for the Series 90-70 PLC consists of one or more units called blocks. Four types of blocks are supported by the Series 90-70 PLC:

Table 2-13. Block Types

Block Type	Programming Language	Size Limit	Number of Parameter Pairs	Notes
LD	Ladder Logic	16 KB in Logicmaster;	n/a	
SFC	Ladder Logic/SFC	16 KB in Logicmaster;	n/a	SFC blocks cannot be used as Interrupt blocks.
PSB	Ladder Logic	16 KB in Logicmaster;	0–7	
External	C	64,000 bytes	0–7	External blocks cannot call any other blocks. External blocks are created using the C Programmer's Toolkit.

- SFC programming is described in detail in the *Series 90 Sequential Function Chart Programming Language User's Manual*, GFK-0854. Sequential Function Chart (SFC) is an IEC-compliant, graphical, state language specifically designed for controlling sequential processes.

- LD (also known as Relay Logic Diagram language) is the language used in the LD Editor window within Control software and in Logicmaster.

- PSB (Parameterized Subroutine Blocks) are LD blocks that have input and output parameters.

Note

Up to 255 blocks can be used. The maximum number of block calls that can be programmed within a given block is 64. The maximum number of programmed calls to a particular block is 255. (A block can be executed any number of times, but there cannot be more than 255 explicit calls to any given block.)

Main Block

When using an LD program there is always a _MAIN block. LD program execution begins with the _MAIN block.

```
        read     inputs
              │
              ▼
         ┌─────────┐
         │ _MAIN   │
         │ BLOCK   │
         └─────────┘
              │
        write    outputs
```
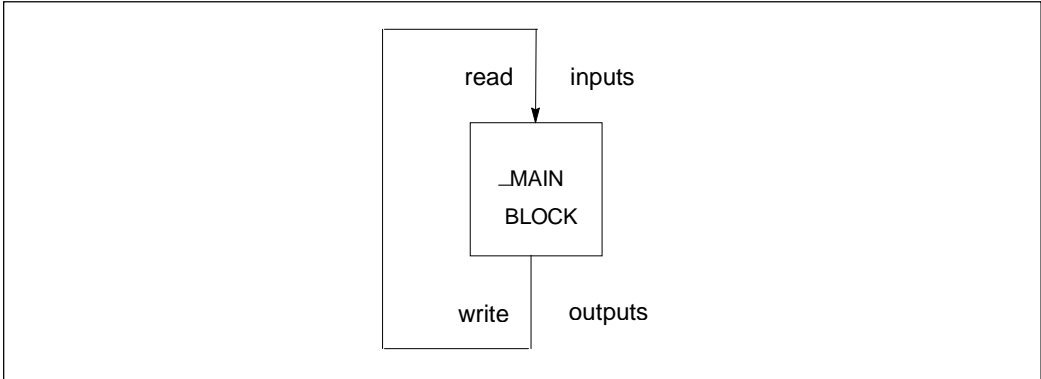
Blocks

Structuring a program as blocks enables you to re-use logic. Logic that needs to be repeated can be entered in a block. Calls would then be made to that block to execute the logic. In this way, total program size is reduced. Dividing a program into smaller blocks also simplifies programming and reduces the overall amount of logic needed for the program.

Examples of Using Blocks

As an example, the logic for an LD program could be divided into three blocks, each of which could be called as needed from the _MAIN block. (A block cannot call the _MAIN block.) In this example, the _MAIN block might contain little logic, serving primarily to sequence the other blocks.
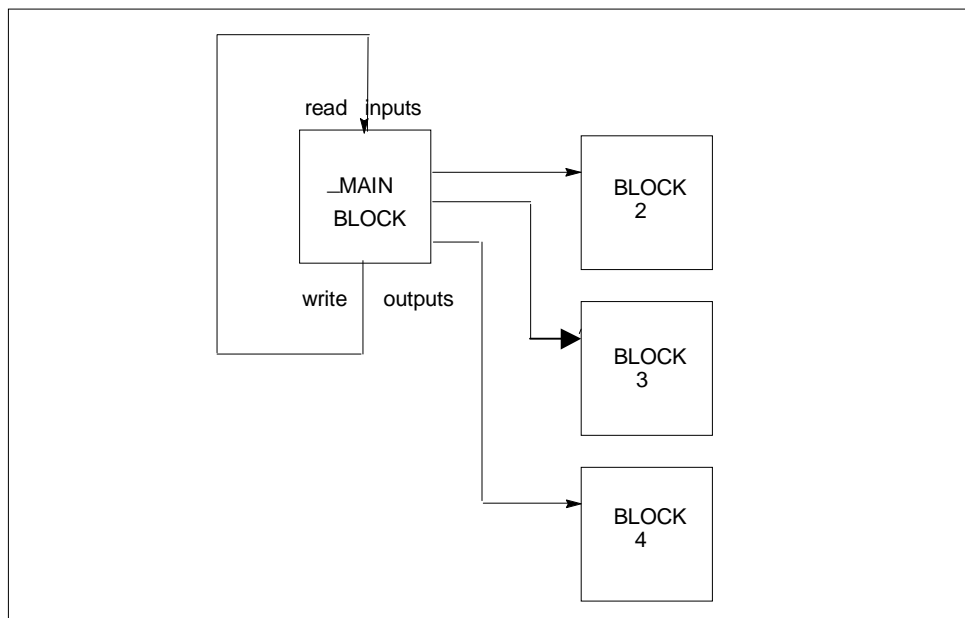
A block can be used many times as the program executes. Logic that needs to be repeated several times in a program could be entered in a block. Calls would then be made to that block to access the logic.

```
                    read  inputs
                        |
                        v
              ┌──────────────┐          ┌──────────┐
              │    MAIN      │─────────>│          │
              │              │─────────>│  BLOCK   │
              │    BLOCK     │─────────>│    2     │
              └──────────────┘          └──────────┘
                        |
                    write  outputs
```
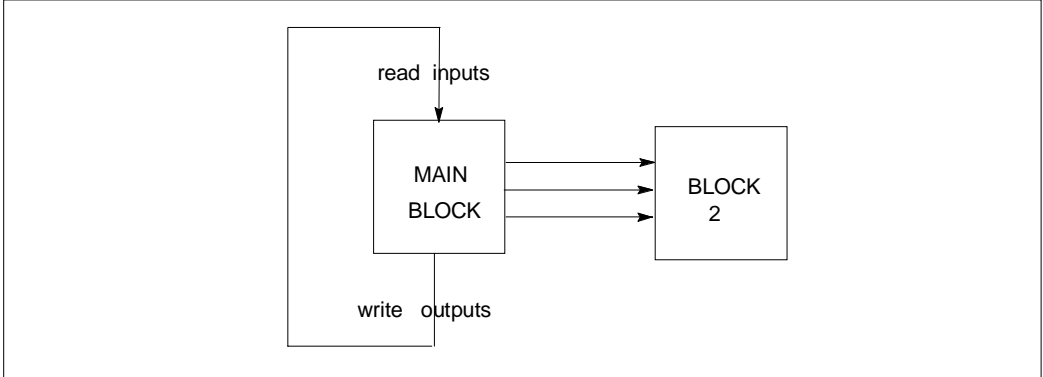
In addition to being called from the _MAIN block, blocks can also be called by other blocks. A block may even call itself.
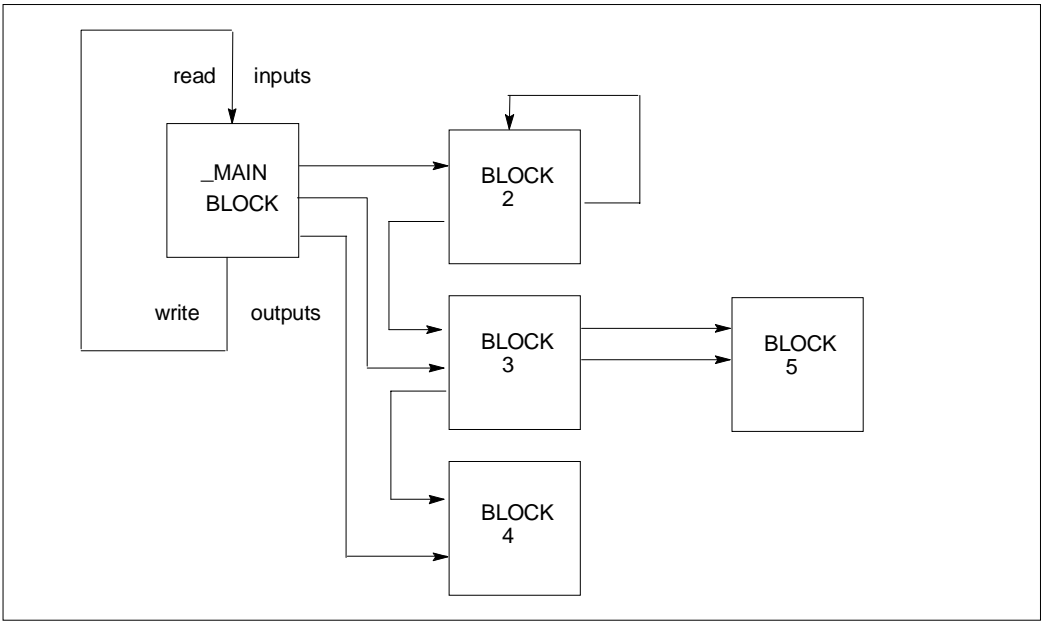
There may be no limit to the number of levels of calls to blocks that your programming software will allow. However, the PLC will only allow a certain number of nested calls before an "Application Stack Overflow" fault is logged and the PLC transitions to Stop/Halt mode. The call depth is guaranteed to be at least four on the 731 and 732 CPUs and eight on all other models. The actual call depth allowed depends on the amount of data (non-Boolean) flow used in the blocks. If less than the 171 word data flow limit is used, then more nested calls may be made. The call level nesting counts the _MAIN block as level 1. The illustration above shows three levels of calls.

Note

Before a block can be used, you need to define it in the block declarations. For information on block declarations, refer to *the Logicmaster 90-70 Programming Software User's Manual* (GFK-0263) .

How Blocks Are Called

A block executes when called from the program logic in the _MAIN block or another block.

```
|   _____
|  |             |
|——|  CALL PBK1  |
|  |_____|
|
|%I00500 %Q00076   _____
|——| |————| |————|             |
|  |             | CALL  PBK2  |
|                |_____|
|
```
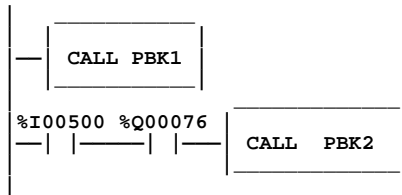
In the example above, PBK1 will always be called. Conditional logic can be used to control calling the block. In order for PBK2 to be called, both input %I00500 and output %Q00076 must be ON.

Blocks and Local Data

Each block in the LD program can have an associated data block. The _MAIN data block is referenced by %P; all other data blocks 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. Appendix C, "Memory Allocation," provides a worksheet for determining the total number of bytes of user data used and how much is still available for the user program.
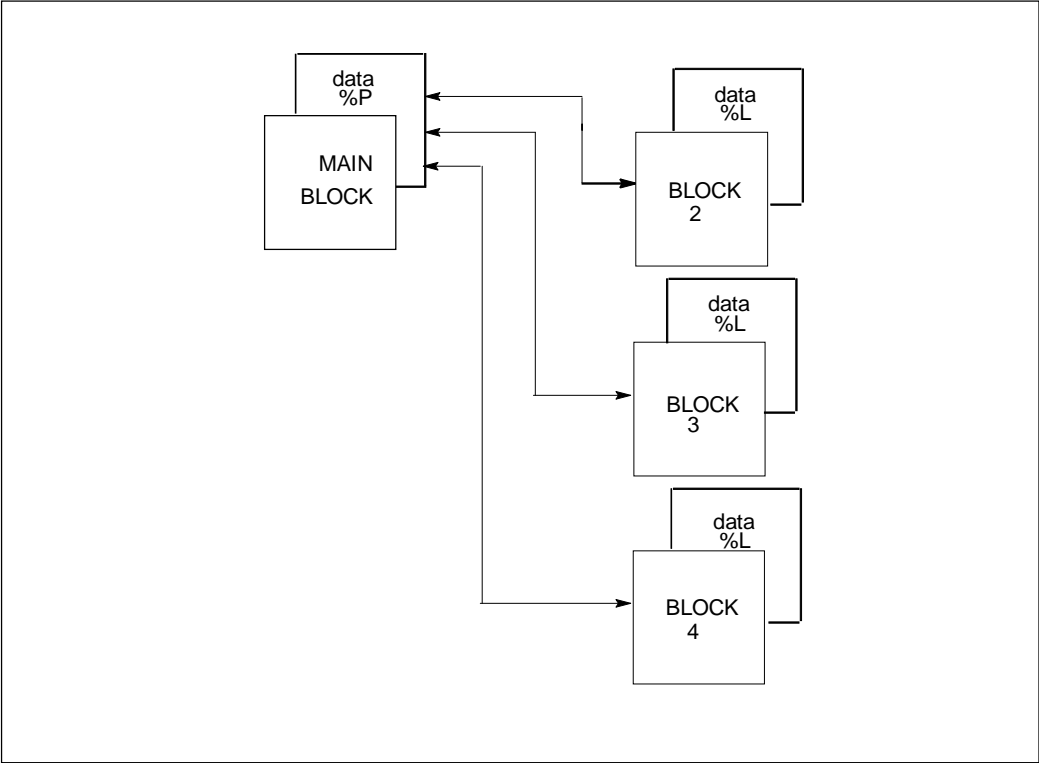
All blocks within the LD program can use data associated with the _MAIN block (%P). Blocks can use their own %L references as well as the %P references that are available to all blocks. The _MAIN block cannot use %L.

Note

External blocks and Parameterized Subroutine Blocks do not have their own %L data; instead, they inherit the %L data of the calling block.

Parameterized Subroutine Blocks

A Parameterized Subroutine Block (PSB) is an optional, user-defined function block, configured with between zero and seven input/output parameter pairs.

As with other blocks, Parameterized Subroutine Blocks can be called by the _MAIN block, other blocks, or itself. The calling block may pass parameters to the Parameterized Subroutine Block. When a Parameterized Subroutine Block is declared, it must be assigned a unique block name along with the number, type, and length of the parameters. Each parameter, other than the Enable and Enable Out parameters, is designated as a BOOL, WORD, or NWORD type, along with a specified length. BOOL lengths range from 1 to 256; WORD and NWORD lengths range from 1 to 1024. Default is one bit for BOOL lengths or one word for WORD and NWORD lengths. In addition, you may also declare an optional three-character formal parameter reference name.

Parameterized Subroutine Blocks and Local Data

Parameterized Subroutine Blocks support the use of %P global data. Parameterized Subroutine Blocks do not have their own %L data, but instead inherit the %L data of the calling block. Parameterized Subroutine Blocks also inherit %S contacts, such as FST_EXE, from the calling block. If %L references are used within a Parameterized Subroutine Block and the block is called by _MAIN, %L references will be inherited from the %P references wherever encountered in the Parameterized Subroutine Block (for example, %L0005 = %P0005).
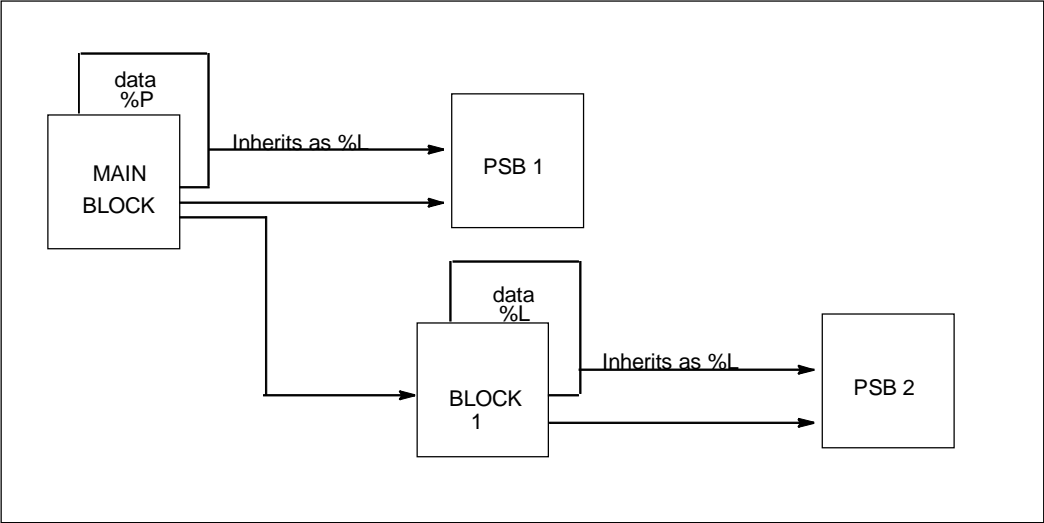
How Parameterized Subroutine Blocks Are Called

A Parameterized Subroutine 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 subroutine named LOAD_41 is executed. The LOAD_41 subroutine block operates on the input data (located at reference addresses %I00100 – %I00111 and %I00001 – %I00016) and produces values in the block of output data (located at reference addresses %T00001 – %T00016, and at register memory addresses %R00200 – %R00201). The logic within the subroutine can also control the OK output of the Parameterized Subroutine Block. This example shows the subroutine CALL instruction as it will appear in the calling block.

```
|         _____
|%I00001 |               |                                                        %Q00001
|—| |————| CALL LOAD_41  |————————————————————————————————————————————————————————( )—
|        | (SUBROUTINE)  |
|        |B012      B016 |
| %I00100 —|ABC        Y1|—%T00001
|        |               |
|        |B016      W002 |
| %I00001 —|X2        Y2 |—%R00200
|        |_____|
|
```
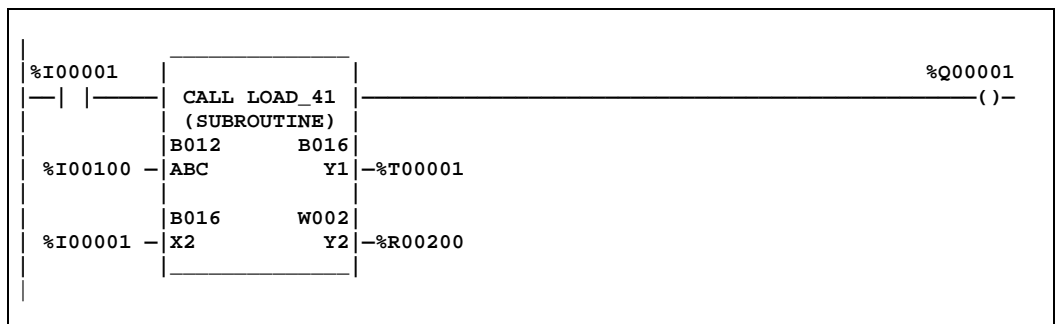
Referencing Formal Parameters Within a Parameterized Subroutine Block

Formal parameters are those parameters used within Parameterized Subroutine Block that are passed from and to the calling block. They are either BOOL, WORD, or NWORD types. NWORD type parameters may be used on any multi-word type operands, but not on discrete types. (An NWORD is a number of words passed into a Parameterized Subroutine Block).

The formal parameters are identified as X input parameters or Y output parameters, followed by the number of the input or output parameter, respectively. For example, X2 indicates the parameter used at location X2 in the parameterized subroutine declaration. The X2 label could be followed by a value of 1 to 16 to the length provided in the subroutine declaration (B016).

Up to seven formal parameter pairs may be declared in a Parameterized Subroutine Block. The formal parameter type, number, and length use the form:

ab[ccc]

where:	**a = X**	denotes an input formal parameter.
	a = Y	denotes an output formal parameter.
	b	is a parameter number between 1 and 7.
	c	is a valid BOOL, WORD, or NWORD index.

The labels X1 through X7 and Y1 through Y7 may be assigned a nickname of up to three characters.

Each parameterized Subroutine Block has a predefined local variable, YO, which the CPU sets to 1 upon each execution of the block. YO can be controlled by logic within the block and provides the output status of the block.

Assigned parameters are PLC references or data flow that pass their address or data into or out of a Parameterized Subroutine Block. An assigned parameter may pass either the value of the data in the assigned parameter (BOOL type parameters) or the address of the assigned parameter (WORD or NWORD type parameters). Assigned parameters are defined in a parameter assignment table.

Restrictions on Formal Parameters within a Parameterized Subroutine Block

In general, formal BOOL parameters are allowed on all contacts, coils, and function block parameters that allow discrete references (%I, %Q, %M, %T, %S, %G, and %U). Formal WORD and NWORD parameters are allowed on all function block parameters that allow register references (%R, %AI, %AQ, %P, %L, and %UR). NWORD parameters are allowed only on multi-word type parameters (that is, DINT, DWORD, or REAL). (An NWORD is a number of words passed into a Parameterized Subroutine Block.)

The following list contains several exceptions and restrictions which have been identified when using formal parameters within a Parameterized Subroutine Block:

1. Transitional contacts, transitional coils, and retentive coils are not allowed with formal parameters. The editor (that is, the editor tool within the programming package) will substitute the non-retentive equivalent of these functions,
   +——(M), +——(SM), and +——(/M) and display an appropriate warning message.

2. Formal BOOL input parameters cannot be used as output parameters on a function block.

3. The DO I/O function is not allowed with formal parameters.

4. Multi-word type function block parameters (that is, DINT, DWORD, or REAL) are only allowed with formal NWORD parameters.

5. Formal parameters are not allowed on the following function block parameters:

Function	Parameter
Service Request (Service Request)	PARMS input parameter.
Communications Request (COMMREQ)	IN input parameter.
DATA_INIT DATA_INIT_COMM DATA_INIT_PID DATA_INIT_ASCII	Q output parameter.

6. A Parameterized Subroutine Block's BOOL type formal parameters may not be passed to another Parameterized Subroutine Block.

7. WORD formal parameters cannot be passed into another Parameterized Subroutine Block's NWORD input parameter.

BIT Type Parameters in PSBs

PSB output parameters of type BIT will affect the transitions in all bytes within the range of the BIT parameter. For example, a BIT parameter of length 4, connected to physical output %Q6, will affect the transitions of all bits within the byte containing %Q1 through %Q8 as well as all bits in the byte containing %Q9 through %Q16.

For arrays of BIT type, each array element is written one bit after the other. Since memory write operations and management of transitional state are done a byte at a time, the transitional status of each element will typically be cleared, except for the final array element. This is not an issue for BIT parameters of length 1, as only a single write operation is performed. Note, however, that as described above, the transition value of other bits within the byte will be affected.

External Blocks

External blocks are created using the C Programmer's Toolkit. Refer to the *C Programmer's Toolkit for Series 90™ PLCs* (GFK-0646) for detailed information regarding external blocks.

How External Blocks Are Called

External blocks are added to a user program by using the Librarian function in your programming software.

An external block executes when called from the program logic in the _MAIN block or from another block. To facilitate the passing and returning of data, an external block may have 0 to 7 parameter pairs.

In the following example, if %I00001 is set, the external block named EXT_11 is executed. The block operates on the input data, located at reference addresses %I00100 – %I00111 and %I00001 – %I00016, and produces values in the block of output data, located at reference addresses %T00001 – %T00016, and at register memory addresses %R00200 – %R00201. The logic within the block can also control the Enable Out output of the external block.

```
 _____
|           _____                                                        |
|%I00001   |            |                                                %Q00001 |
|—| |——————| CALL EXT_11|————————————————————————————————————————————————( )—   |
|          | (EXTERNAL) |                                                        |
|          |            |                                                        |
| %I00100 –|X1        X1|–%T00001                                                |
|          |            |                                                        |
|          |            |                                                        |
| %I00001 –|X2        X2|–%R00200                                                |
|          |_____|                                                        |
|_____|
```
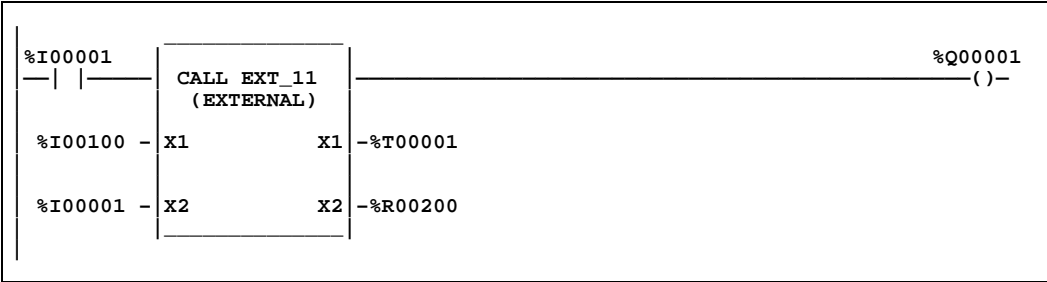
Note

Unlike other block types, external blocks cannot call any 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 the calling block. External blocks also inherit %S contacts, such as FST_EXE, from the calling block. 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).
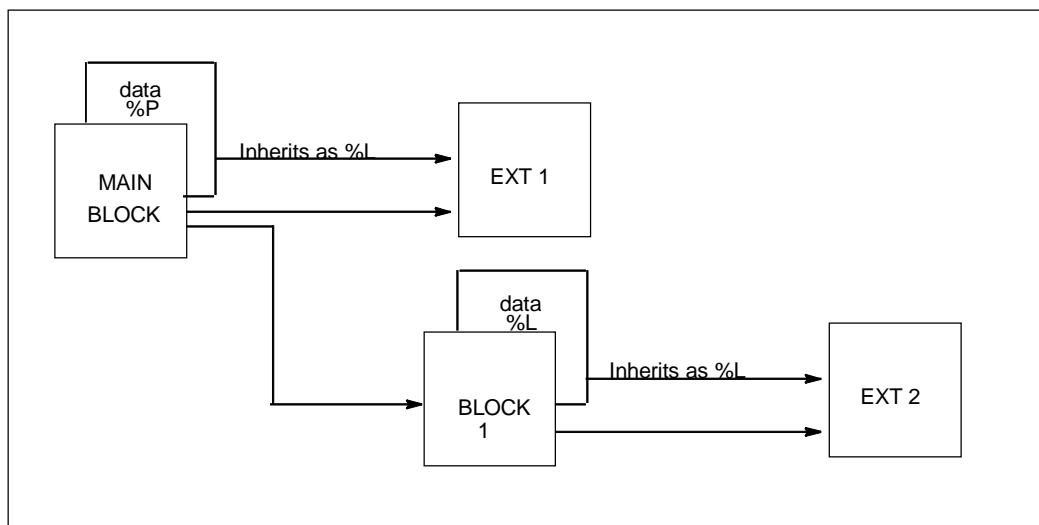
Local Data Initialization

When an external block is stored to the PLC, a copy of most of its internal data is saved. Global and static initialized data are saved, but if static variables are declared without an initial value, the initial value is undefined and must be initialized by the C application. (Refer to the "Global Variable Initialization" and "Static Variable" parts in Section 6 of Chapter 2 of the *C Programmer's Toolkit for Series 90™ PLCs* (GFK-0646). This data is used to re-initialize the block's data area whenever the PLC transitions from Stop to Run.

External/standalone programs do not use %L data, but the internal data they use is somewhat similar to local data, as discussed above. Internal data used in a standalone program should not be confused with %L data.

Standalone C Programs

Like external blocks, standalone programs are developed using the C Programmer's Toolkit. Unlike external blocks, however, standalone C Programs can be up to 564 KB in size. Standalone C programs cannot call other standalone programs, nor can they call blocks within an LD program. Similarly, blocks within an LD program cannot call a standalone program. Instead, standalone C programs are scheduled for execution using one of several possible program scheduling modes, described in section 4 of this chapter.

Note

For information on adding C programs to your folder, refer to the Logicmaster 90-70 Programming Software User's Manual (GFK-0263) .

A maximum of 16 standalone programs can be used at one time. If an LD program is used, only 15 standalone programs are allowed, for a total of 16 programs.

Note

Since standalone C programs are truly separate programs, they do not have access to memory types local to an LD program (%L and %P). The internal or local data used in a standalone C program should not be confused with %L and %P data.

Data Encapsulation

Each standalone C program is provided with a means of obtaining its own local copy of user data references. Instead of operating on the global set of user references, each standalone program can operate on its own local set of data. This feature is supported through the use of an input/output specification.

The following steps occur when using an I/O specification with a standalone C program:

1. When the program is scheduled for execution, any corresponding input specification is copied from the global user reference data area(s) to an area local to the program.

2. As the program executes, it can operate on its local set of input and output data. Any interruptions during the execution phase will not affect this program's local copies of input or output data.

3. When the program completes, its local output specification is copied back to the specified set of user data reference areas.

Two particular concerns are addressed by using an input/output specification. First, if a program is suspended mid-execution and an output scan is performed before execution is resumed, output values will remain consistent since the output scan values are obtained from the user data reference locations. Second, if a program is interrupted mid-execution by another program, the first program is unaffected by changes to global data caused by the second program, since it has its own local copy of data.

Another benefit to using an input specification is to provide a more accurate sampling of input values. If a program's execution is postponed due to higher priority programs, the input specification may provide the program with a set of data that more accurately represents the state of that data when the program was scheduled since global user reference data may have been modified by higher priority programs or by the scanning of input values.

Input/Output Specifications

Unlike external blocks, standalone C and LD programs cannot have any input or output parameters. Standalone C programs may utilize an input/output specification which lists a maximum of eight input and eight output ranges. The input ranges will be copied to the program at the start of program execution. The output ranges are copied from the standalone C program to the global reference on the completion of program execution. Note that this differs from external block parameters which are passed by reference, not by value. This operation is especially important for programs which may be time-sliced over multiple sweeps, which can occur when using Microcycle Sweep mode[1].

```
┌─────────────────┐
│     Caution     │
└─────────────────┘
```

When the PLC runs in Microcycle Sweep mode, programs can be suspended in the middle of execution, possibly in the middle of a line of C code or in the middle of a rung of logic or function block. If the program uses global references such as %Q, %R, etc., a possibly inconsistent set of reference values may be present at the time an Interrupt program or output scan occurs. This inconsistency could even be within a given reference value if the value is not accessed according to its type.

---

[1] Incoherent data can result if a program uses global data and is suspended across multiple sweeps. The data referenced will be from two successive sweeps. Although data cannot be incoherent within a byte or word, global data should only be accessed using its basic type (byte, word, etc.), otherwise incoherency can apply to individual elements as well.

To illustrate the possibility of inconsistent output data, consider the following example program which updates a given output value in several locations. For this example, assume that the user has configured %AQ1 as an output specification from the second program, and that user reference %AQ1 contains the value 0 when the two programs begin execution.

Differences Between Accessing Global and Local (Internal) Data

Program Accessing Global Data		**Program Using I/O Specification**	
<program begins>		<program begins>	
%AQ1 = 0	An output scan occurring after this line of code would output a 0 to AQ1.	LOC[1] = 0	An output scan occurring after this line of code would output a 0 to AQ1.
%AQ1 = %AQ1 + 2	An output scan occurring after this line of code would output a 2 to AQ1.	LOC[1] = LOC[1]+2	An output scan occurring after this line of code would output a 0 to AQ1.
%AQ1 = %AQ1 + 12	An output scan occurring after this line of code would output a 14 to AQ1.	LOC[1]=LOC[1] + 12	An output scan occurring after this line of code would output a 0 to AQ1.
%AQ1 = %AQ1 / 2	An output scan occurring after this line of code would output a 7 to AQ1.	LOC[1]=LOC[1]/2	An output scan occurring after this line of code would output a 0 to AQ1.
<program completes>		<program completes>	
		The PLC copies the output specification from a local area back to the global areas when the program completes.	
Further output scans will output a 7 to AQ1.		Further output scans will output a 7 to AQ1.	

Note

Remember that standalone C programs are truly separate programs. They do not have access to memory types local to an LD program (%L and %P). The internal or "local" data used in a standalone C program should not be confused with %L and %P data.

Standalone C Programs and Local Data

Standalone C programs do not have a local data copy provided by the PLC. Similarly, they are not able to access %P memory of an LD program, nor can they access any %L memory associated with a block within an LD program. Standalone C programs do have local data that is declared within the C source file(s) used to create the standalone C program. Refer to the *C Programmer's Toolkit User's Manual* (GFK-0646) for further information.

Local Data Initialization

When a standalone C program is stored to the PLC, a copy of its internal data is saved. This data is used to re-initialize the program's data area whenever the PLC transitions from Stop to Run.

Referencing I/O Specification Data Within a Standalone C Program

Several new C macros used to define and access input and output specification data are defined for standalone C programs. Refer to the *C Programmer's Toolkit User's Manual* (GFK-0646, revision C or later) for information regarding referencing I/O specification data within a standalone C program.

Data Coherency of I/O Specifications

Since standalone C programs can be interrupted by other programs and Interrupt blocks, data incoherency within an I/O specification can occur. Each individual I/O specification is limited to 2048 bytes, for a maximum of 16KB of input data and 16KB of output data. The 90-70 PLC will ensure the following:

- Each byte within an individual I/O specification is coherent with respect to that individual specification.

- If the total length of all input specifications is no more than 2048 bytes, the entire input specification will be coherent.

- If the total length of all output specifications is no more than 2048 bytes, the entire output specification will be coherent.

If the total length of an input or output specification exceeds 2048 bytes, groups of individual specifications whose combined lengths do not exceed 2048 bytes will be coherent. The following table indicates coherency in this case.

Table 2-14. Coherency of I/O Specification

I/O specification	Length (bytes)
1	2000
2	48
Interrupts may occur	
3	1024
Interrupts may occur	
4	1026
Interrupts may occur	
5	10
6	20
7	20
8	20

Using LD vs. Standalone C Programs

Several options need to be considered when determining which type of program is to be used. The following list summarizes many of the features supported in each of the types of programs:

- Interrupt Blocks are preferred over standalone programs when interrupt latency is a concern. The overhead to process an Interrupt Standalone program is much larger than that of an Interrupt block.

- LD programs may operate only on global user reference data. This can introduce data coherency problems when the LD program is run in Microcycle Sweep mode and the program is suspended over multiple sweeps.

- Standalone C programs incur an 8K overhead per program.

- An LD program is preferred over a standalone C program when you are using large amounts of Boolean instructions. LD programs are better suited for relay-type logic than are standalone C programs.

Table 2-15. LD vs. Standalone C Program Tradeoffs

	Programming Language	Program Size Limit	Data Types Accessible	Local Data Size	Scheduling Modes	Data Encapsulation	Block Types Supported
LD	Ladder Logic	smaller of 544K or available memory size, organized into 16K blocks.	All	8k %P, 8K %L per block	All*	No	LD SFC PSB External
Standalone	C, using the C Programmer's Toolkit.	smaller of 564K or available memory size	All except %P, %L	unlimited, counts as part of program size	All *	Yes	n/a

*Using Microcycle Sweep mode with an LD program is not recommended. LD programs always operate on global data directly, which can lead to inconsistent output values if the LD program is suspended mid-execution. Refer to section 4 of this chapter for more information on Microcycle Sweep mode.

Differences in Operation: LD and Standalone C Programs

Retentiveness of Data

When only standalone C programs are used, the retentive nature of data is based solely on memory type since there are no coil instructions. In this case %Q and %M are retentive. If both LD and standalone C programs are used, the retentive property of memory types is driven by their use in the LD program. For more information about retention of logic and data, refer to the "Retentiveness of Logic and Data" discussion in section 2 of this chapter. For more information on retentive properties of specific memory types, refer to the Table 2-3.

Global Data

LD programs only have access to global data areas since they do not have the ability to use input/output specifications. This can lead to inconsistent output values if an LD program is used in Microcycle Sweep mode.

Interrupt Execution

Interrupt blocks within the LD program have the highest priority in the system. In addition, they cannot be preempted, while standalone C programs can be.

Queuing of Interrupts

There are differences between programs and blocks in the queuing of additional interrupts. Refer to the "Interrupt Handling" on page 2-64 and the related sections that follow for detailed information.

System Status References

The following differences exist when using System Status References* (called "Convenience References" in previous editions of 90-70 Reference manual):

• The reference FST_EXE is not available to standalone C programs.

• The reference FST_SCN does not refer to %S0001 within standalone C programs. Instead, a macro is provided by the C Toolkit to provide identical functionality.

*For information on System Status References, refer to Table 2-9 as well as section 2 of this chapter.

Section 4: PLC Sweep Modes and Program Scheduling Modes

Normal Sweep Mode

In Normal Sweep mode, each PLC 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 a Limited or Run-to-Completion mode. Normal Sweep is the most common sweep mode used for PLC applications.

The following figure illustrates three successive PLC 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.

a45529

SWEEP n	SWEEP n+1	SWEEP n+2
HK	HK	HK
INPUT	INPUT	INPUT
LOGIC	LOGIC	LOGIC
OUTPUT	OUTPUT	OUTPUT
PRG	PRG	PRG
SYS	SYS	SYS
BG	BG	BG

Abbreviations:

HK = Housekeeping
PRG = Programmer Window
SYS = System Communications Window
BG = Background Window
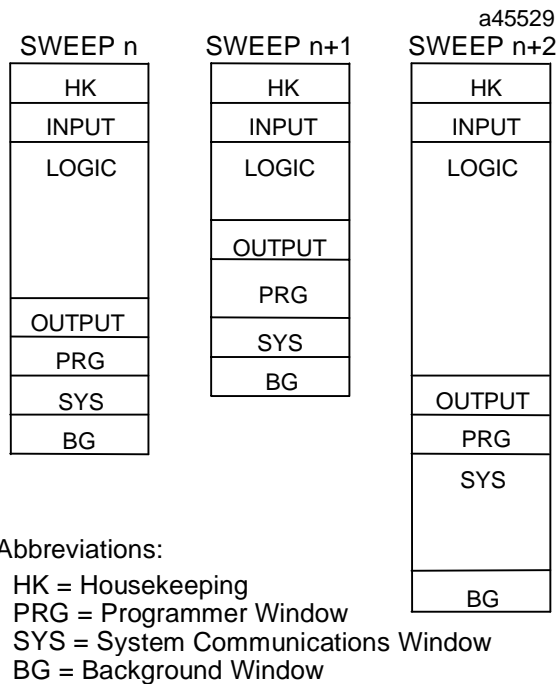
Figure 2-3. Typical Sweeps in Normal Sweep Mode

Constant Sweep Mode

In Constant Sweep mode, each PLC sweep begins at a specified Constant Sweep time after the previous PLC sweep began. The Logic Window is executed in its entirety each sweep. If there is sufficient time at the end of the sweep, the PLC will alternate among the Programmer Communications, System Communications, and Background Windows, allowing them to execute in Run-to-Completion mode 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 will terminate when the overall PLC 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 are updated at constant intervals.

The value of the Constant Sweep timer can be configured to be any value from 3 to 255 milliseconds. The Constant Sweep timer value may also be set and Constant Sweep mode may be enabled or disabled by your 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 exchange 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 PLC constant sweep mode and setting the CPU watchdog timeout.

If the PLC sweep exceeds the Constant Sweep time in a given sweep, the PLC places an oversweep alarm in the PLC fault table and sets the OV_SWP (%SA0002) status reference at the beginning of the next sweep. The OV_SWP status reference is reset when the time of the last sweep does not exceed the Constant Sweep timer or the PLC is not in Constant Sweep mode. 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 PLC 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.
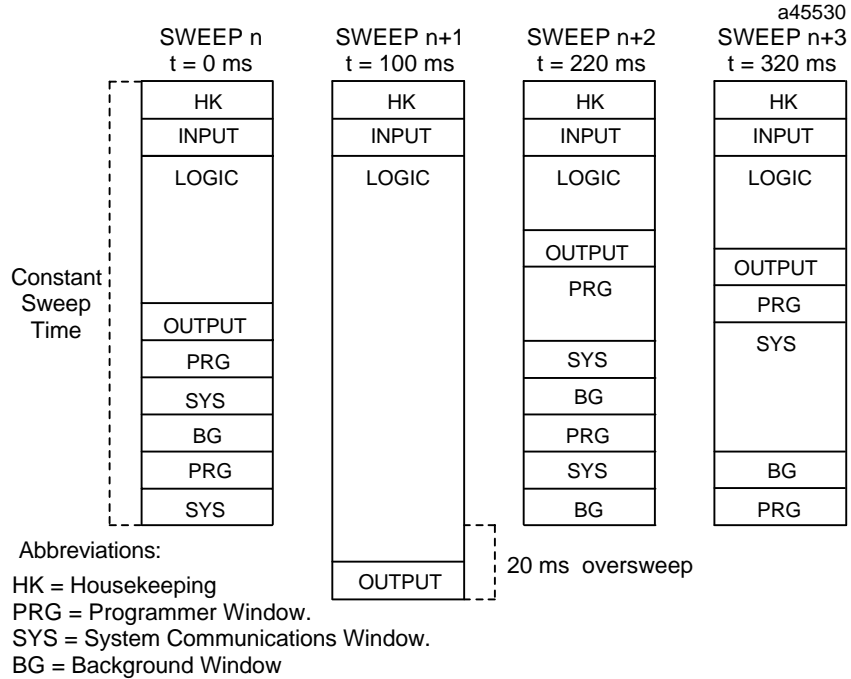
Figure 2-4. Typical Sweeps in Constant Sweep Mode

Constant Window Mode

In Constant Window mode, each PLC sweep can consume a variable amount of time. The Logic Window is executed in its entirety each sweep. In this mode, the PLC will alternate among the three windows, allowing them to run in a Run-to-Completion mode for a time equal to the value set for the Constant Window timer. The overall PLC 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 5 to 255 milliseconds. The Constant Window timer value may also be set by your programming software or by the user program using Service Request functions #3, #4, and #5.

The following figure illustrates three successive PLC 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.

Abbreviations:

HK = Housekeeping
PRG = Programmer Window
SYS = System Communications Window
BG = Background Window
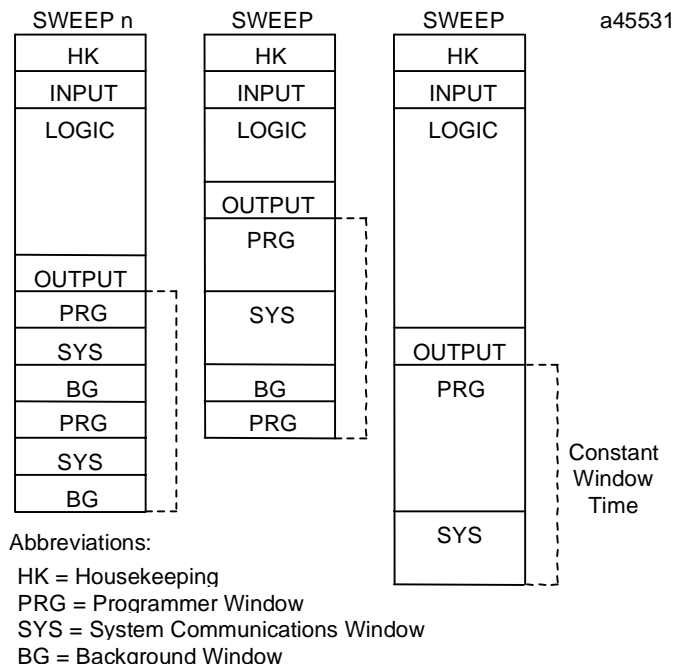
Figure 2-5. Typical Sweeps in Constant Window Mode

Microcycle Sweep Mode

In Microcycle Sweep mode, each PLC sweep begins at an absolute time— which is a multiple of the base cycle time—relative to the Stop-to-Run transition of the PLC. The base cycle time specifies how long each sweep should take (similar to the Constant Sweep time in Constant Sweep mode). The user programs are scheduled for execution each sweep based on their period and may execute in a time-sliced fashion over multiple PLC sweeps. The PLC will alternate between the Communications and Background Windows, allowing them to run in a Run-to-Completion mode until it is time for the next sweep to begin.

Microcycle Sweep mode can be used to allow some programs to execute more often than others. This allows more processing time to be applied to the more important or more time-critical tasks. Microcycle Sweep mode also allows programs to execute more in line with the time when their inputs are available.

Although Microcycle Sweep mode has a fixed sweep time, it is significantly different from Constant Sweep mode. First, user programs do not necessarily execute in their entirety each sweep. In order to maintain the base cycle time and the Communications and Background Window times, user programs may be suspended during execution and resumed the following PLC sweep. Also, additional sweep time due to an oversweep condition in a given sweep causes the next sweep to be

shortened by the oversweep time. In this way each PLC sweep (with the exception of sweeps which follow an oversweep condition) begins at an absolute time relative to the Stop-to-Run transition of the PLC. Finally, the Communications and Background Windows are guaranteed to run for at least the specified Window time each sweep. The Logic Window will be suspended, if necessary, to guarantee that the Communications and Background Windows get to run for the specified Window time.

The base cycle time and the window timer value can be configured with a base cycle time of between 5 and 2550 milliseconds. The Constant Window timer can be any value from 5 to 255 milliseconds. The base cycle time and Constant Window timer may also be set while the PLC is in Stop mode. The base cycle time and window timer cannot be changed while the PLC is in Run mode.

In Microcycle Sweep mode, Periodic programs execute on a priority basis. Periodic programs have priority inverse to their period (smallest period has highest priority).   Refer to the "User Program Execution" discussion later in this section for more information on Periodic programs and their execution.

If the PLC sweep exceeds the base cycle time in a given sweep, the PLC places an oversweep alarm in the PLC fault table and sets the OV_SWP (%SA0002) status reference at the beginning of the next sweep. The OV_SWP status reference is reset when the last sweep time does not exceed the base cycle time. Sweep time due to an oversweep condition in a given sweep causes the next sweep to be shortened by the oversweep time.

The following figure illustrates three successive PLC sweeps in Microcycle Sweep mode with a base cycle time of 100 milliseconds. Note that the sweep time is constant and the Communications and Background Windows are guaranteed to run for the configured window timer. In sweep n and sweep n+1, the Logic finishes early; an I/O-Triggered program can execute during that time. In sweep n+2, the Logic Window is not complete and is suspended so that the Communications and Background Windows can run for the specified window time. In each case the logic window stays open for the entire time allowed for logic.

a45532

	SWEEP n t = 0 ms		SWEEP n+1 t = 100 ms		SWEEP n+2 t = 200 ms
	HK		HK		HK
	INPUT		INPUT		INPUT
Time Allowed for Logic	LOGIC		LOGIC		LOGIC
	OUTPUT		OUTPUT		OUTPUT
	PRG		PRG		PRG
	SYS		SYS		
Window Timer	BG		BG		SYS
	PRG		PRG		
	SYS		SYS		BG

Base Cycle Time (bracket spanning all sweeps on left)

Abbreviations: HK = Housekeeping
PRG = Programmer Window.
SYS = System Communications Window.
BG = Background Window
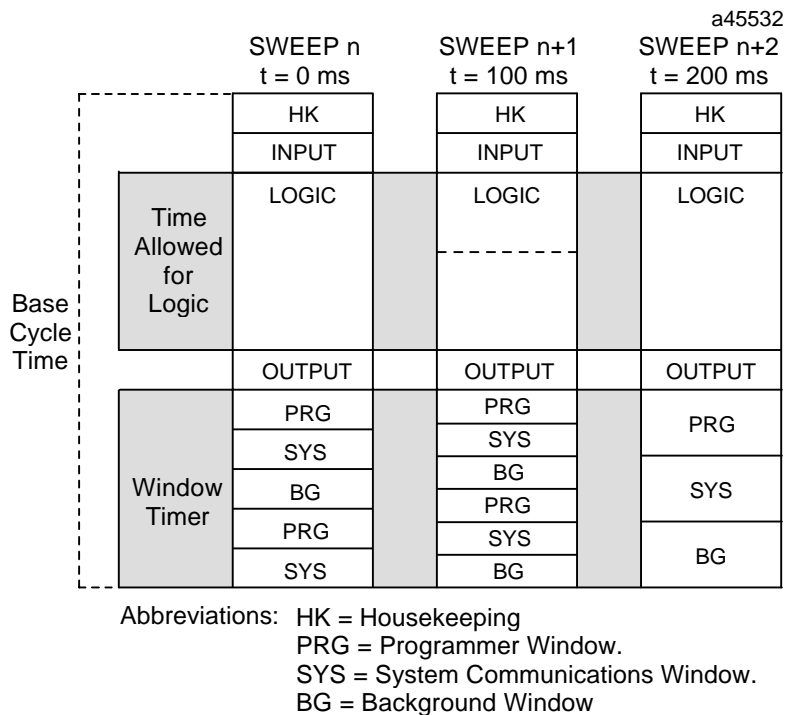
Figure 2-6. Typical Sweeps in Microcycle Sweep Mode

Note

Run Mode Store of logic is not supported in Microcycle Sweep mode. Also, the Single Sweep Debug feature is not supported in Microcycle Sweep mode.

Microcycle Sweep Mode Output Scan Estimation

Microcycle mode was a new CPU sweep mode beginning with Release 6.00 CPUs. In this sweep mode, each PLC sweep begins at an absolute time relative to the Stop-to-Run transition of the PLC. In order to meet the deadline for the start of the next sweep, the CPU must estimate the time required for the Output Scan of the current sweep.

Output Scan Estimation for Pre-Release 7.00 CPUs

For Pre-Release 7.00 CPUs, the estimate of the Output Scan is based on the previous sweep's Output Scan time. For example, if the actual output scan time for sweep N is 15.3 ms, then the CPU will allot 15.3 ms to execute the output scan for sweep N+1.For the first PLC scan, Pre-Release 7.00 CPUs will estimate the Output Scan to be one-third of the configured base cycle time. For example, if the base cycle time is configured to be 60 milliseconds, then the Output Scan for the first PLC scan will be estimated to require 20 milliseconds. In this example, if the Output Scan does not actually require 20 milliseconds, the time allotted for the Logic Window for the first PLC

scan will be shorter than the Logic Window time for the remaining sweeps. When programming PLC logic for first scan, ensure that the logic to be performed in a given program will complete prior to the next execution time for that same program.

Output Scan Estimation for Release 7.00 and Later CPUs

For Release 7.00 and later CPUs, the actual amount of output data that is scheduled to be scanned for the current PLC sweep is used to estimate the Output Scan time. This estimate is based on empirically measured times for the type and amount of output data to be scanned. This results in a fairly accurate estimate of the output scan time which is designed to maximize the amount of time spent in the Logic Window while maintaining the user-configured Communications Window time. No special consideration of the logic execution time for first sweep is required.

Program Scheduling Modes

Each user program in the 90-70 PLC can execute, subject to sweep mode restrictions, in one of four program scheduling modes. This section will briefly describe the following four available program scheduling modes:

- Ordered

- Timed

- I/O-Triggered

- Periodic

Ordered   Ordered programs are executed in the Logic Window with all other Ordered programs. Ordered programs are executed once per sweep in the sequence in which they are declared in the programming software. Ordered programs are not supported in Microcycle Sweep Mode.

Timed   Timed programs are scheduled to execute on a specified time interval with an initial delay (if specified) applied on Stop-to-Run transition of the PLC. Timed programs are scheduled to execute on a priority basis during any phase of the PLC sweep. Timed programs are not supported in Microcycle Sweep mode.

I/O-Triggered   I/O-Triggered programs are scheduled to execute on the receipt of a configured I/O Interrupt. I/O-Triggered programs are scheduled to execute on a priority basis during any phase of the PLC sweep when the PLC is in Normal Sweep, Constant Sweep, or Constant Window Sweep mode.

In Microcycle Sweep mode, I/O-Triggered programs are scheduled to execute on a priority basis in the Logic Window. In this case, the execution of I/O-Triggered programs may be time-sliced over multiple sweeps.

Periodic   Periodic programs are scheduled for execution based on the user-configured period for the program and are scheduled to execute in the Logic Window with all other Periodic programs. Periodic programs execute on a priority basis relative to all other programs and may be time-sliced over multiple sweeps. Periodic programs are only supported in Microcycle Sweep mode.

Choosing PLC Sweep and Program Scheduling Modes

The table shown below indicates the availability of each program scheduling mode in each of the available PLC sweep modes.

Table 2-16. Available Program Scheduling Modes in Each PLC Sweep Mode

Sweep Mode	Program Scheduling Mode				Interrupt Blocks
	Ordered	**Timed**	**I/O Triggered**	**Periodic**	
Normal	Yes	Yes	Yes	No	Yes
Constant	Yes	Yes	Yes	No	Yes
Constant Window	Yes	Yes	Yes	No	Yes
Microcycle	No	No	Yes *	Yes	Yes

* Executes in Logic Window only.

User Program Execution

User Program Priorities

The priority of a user program specifies its priority relative to other programs. Higher priority programs execute before lower priority programs. If two or more programs with the same priority are scheduled at the same time, the order of execution is undefined. Programs can be suspended in the middle of execution by higher priority programs and Interrupt blocks.

Ordered programs all have the same priority and are executed in the order which you specify in the programming software. Ordered programs have lower priority than Timed programs, I/O-Triggered programs, and Interrupt blocks.

Periodic programs have priority inverse to their period (smallest period has highest priority). The order of execution of Periodic programs with the same period is undefined. Periodic programs have lower priority than Timed programs, I/O-Triggered programs, and Interrupt Blocks.

Timed and I/O-Triggered programs have higher priority than Ordered and Periodic programs. The priority of a Timed or I/O-Triggered program specifies its priority relative to other Timed and I/O-Triggered programs. The priority range 10–99 (10 being the highest priority) is reserved for Timed and I/O-Triggered programs which can run during any phase of the PLC sweep (that is, not restricted to running in the Logic Window). Timed and I/O-Triggered programs operate this way when the PLC is running in Normal Sweep, Constant Sweep, or Constant Window mode. In Microcycle Sweep mode, I/O-Triggered programs are executed in the Logic Window, and priorities of 100–109 (100 being the highest priority) are reserved for this mode.

Timed and I/O Interrupt blocks have the highest priority of any user logic.

Table 2-17. Priority Values for Timed and I/O-Triggered Programs

Scheduling Mode	Sweep Mode	
	Normal Sweep Constant Sweep Constant Window	**Microcycle Sweep**
Ordered *	Executed in order declared in the programming software.	Not supported.
Periodic *	Not supported.	Smallest period has highest priority.
I/O-Triggered	10–99	100–109
Timed	10–99	Not supported.

* Ordered and Periodic scheduling modes have lower priority than Timed and I/O-Triggered scheduling modes.

User Program Execution in Normal Sweep, Constant Sweep, and Constant Window Modes

In Normal Sweep, Constant Sweep, and Constant Window modes, the 90-70 PLC can execute Ordered, Timed, and I/O-Triggered programs as well as Timed and I/O Interrupt blocks.

Ordered programs execute in their entirety once per sweep in the Logic Window. The programs execute in the order in which they are declared in the programming software. The input specification is copied prior to execution of the program, and the output specification is copied upon completion of the program. In this way, the output of one program can be used as input for the next, if desired.

The following figure depicts two Ordered programs (A and B) executing in a typical PLC sweep in Normal Sweep mode.
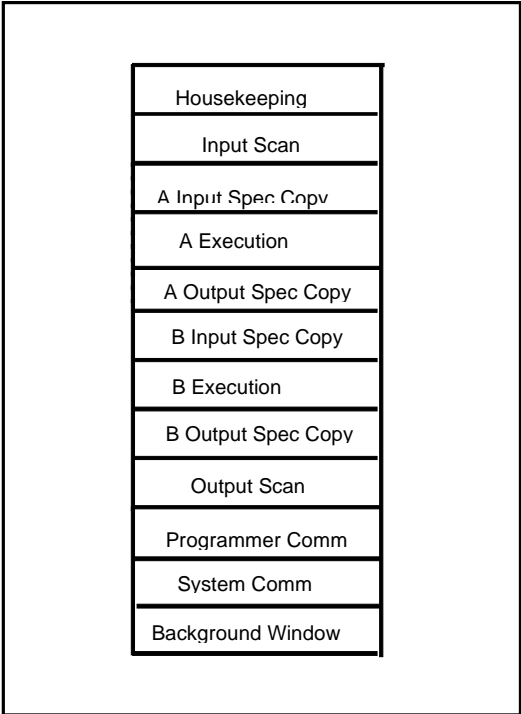
Figure 2-7. Ordered Program Execution Sequence

In Normal Sweep, Constant Sweep, and Constant Window mode, Timed and I/O-Triggered programs execute during any phase of the PLC sweep. These programs will preempt the execution of Ordered programs and lower priority Timed and I/O-Triggered programs. The input specification is copied at the time the program is scheduled to execute (that is, when the time interval expires or the I/O Interrupt occurs). The output specification is copied upon completion of the program.

Timed and I/O Interrupt blocks execute during any phase of the PLC sweep. These blocks will preempt the execution of all programs and have the highest priority of any user logic in the PLC. Timed and I/O Interrupt blocks do not have an input or output specification copy.

The following figure depicts 2 Ordered programs (A and B) , an I/O-Triggered program (C) with priority 10, a Timed program (D) with priority 20, and an I/O Interrupt block all executing in a typical PLC sweep in Normal Sweep mode.
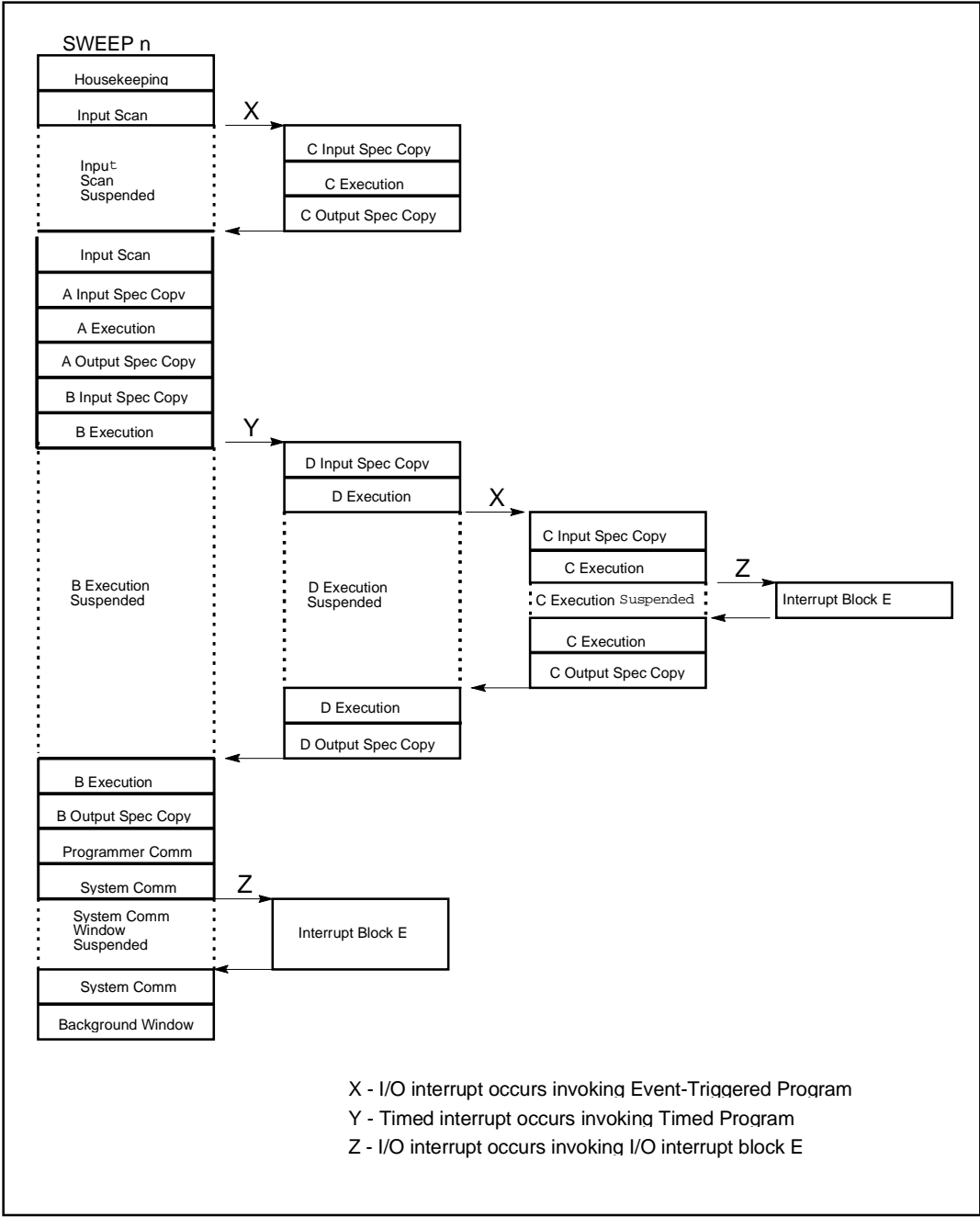
Figure 2-8. Ordered, Timed, I/O-Triggered and Interrupt Block Execution Sequence

User Program Execution in Microcycle Sweep Mode

In Microcycle Sweep mode, the 90-70 PLC can execute Periodic and I/O-Triggered programs as well as Timed and I/O Interrupt blocks.

Periodic programs execute in the Logic Window. These programs are scheduled to execute based on the program's period. For example, a program with a period of 1 will be scheduled to execute every PLC sweep and a program with a period of 2 will be scheduled to execute every other PLC sweep. Periodic programs have priority inverse to their period (smallest period has highest priority). These programs are subject to time-sliced execution over multiple sweeps based on the time available to the Logic Window. Unlike Ordered programs, the input specification is copied at the beginning of the Logic Window for all Periodic programs that are scheduled to begin execution in a given sweep. In other words, all input specification copies will occur for Periodic programs before any of the Periodic programs begin or continue executing. The output specification is copied upon completion of the program.

The following figure depicts two Periodic programs (A and B) executing in a typical PLC sweep in Microcycle Sweep mode.

| Housekeeping |
| Input Scan |
| A Input Spec Copy |
| B Input Spec Copy |
| A Execution |
| A Output Spec Copy |
| B Execution |
| B Output Spec Copy |
| Output Scan |
| Programmer Comm |
| System Comm |
| Background Window |
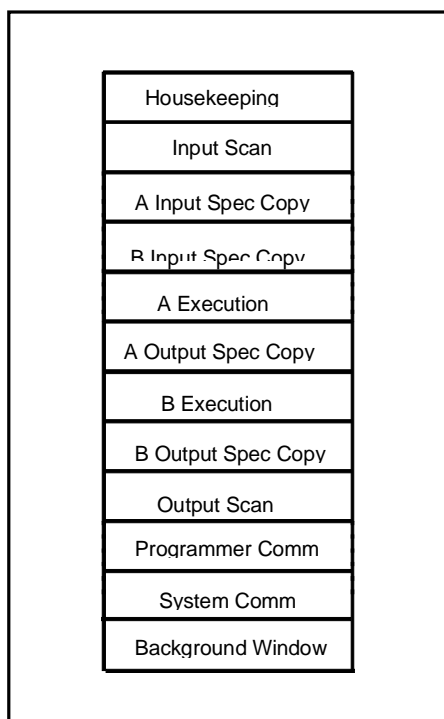
Figure 2-9. Periodic Program Execution Sequence

Unlike other sweep modes, I/O-Triggered programs execute in the Logic Window only when the PLC is in Microcycle Sweep mode. If the I/O Interrupt occurs during or prior to the end of the Logic Window, the I/O-Triggered program will be scheduled to execute in the Logic Window of the current PLC sweep. Otherwise, it will be scheduled to execute in the Logic Window of the next PLC sweep. I/O-Triggered programs will preempt the execution or resumption of Periodic programs and lower priority I/O-Triggered programs. These programs are subject to the same time-sliced execution over multiple sweeps as Periodic programs, based on the time available to the Logic Window. The input specification for an I/O-Triggered program is copied at the time the

program is scheduled to execute (that is, when the I/O interrupt occurs) not at the beginning of the Logic Window as with Periodic programs. The output specification is copied upon completion of the program.

Timed and I/O Interrupt blocks execute during any phase of the PLC sweep when the PLC is in Microcycle Sweep mode. These blocks will preempt the execution of all programs and have the highest priority of any user logic in the PLC. Timed and I/O Interrupt blocks do not have an input or output specification copy.

The following figure depicts two Periodic programs (A and B) and one I/O-Triggered program (C) executing in two successive Microcycle Sweeps. Periodic programs A and B both have a period of 1.

SWEEP n

Housekeeping
Input Scan
A Input Spec Copy
B Input Spec Copy
A Execution
A Output Spec Copy
B Execution
B Output Spec Copy
Output Scan
Programmer Comm
System Comm

X

System Comm Window Suspended
System Comm
Background Window
Programmer Comm

C Input Spec Copy

Programmer Communications window
suspended due to window time expiring

SWEEP n+1

Housekeeping
Input Scan
A Input Spec Copy
B Input Spec Copy
C Execution
A Execution

X

A Execution Suspended
C Execution
A Execution
A Output Spec Copy
B Execution
B Output Spec Copy
Output Scan
Programmer Comm
System Comm

C Input Spec Copy

System Communications Window
suspended due to window time expiring

X - I/O interrupt occurs invoking Event-Triggered program

Figure 2-10. Periodic and I/O-Triggered Execution Sequence

Global Data in Microcycle Sweep Mode

Incoherent data can result if a program that uses global data (%R, %I, %Q, etc.) is suspended across multiple sweeps. The data referenced will be from two successive sweeps. Although data cannot be incoherent within a byte or word, global data should only be accessed using its basic type (byte, word, etc.); otherwise, incoherency can apply to individual elements as well. If possible, the input and output specifications should be used to access and update global data areas.

Interrupt Handling

There are two types of interrupts available for user program handling in the 90-70 PLC.

I/O Interrupts These interrupts are generated by 90-70 I/O modules to indicate discrete input state changes (rising/falling edge), analog range limits (low/high alarms), high speed signal counting events, and interrupts from 3rd party VME modules.

Timed Interrupts These interrupts are generated by the 90-70 PLC CPU based on a user-specified time interval with an initial delay (if specified) applied on Stop-to-Run transition of the PLC.

Both of these types of interrupts may invoke a user program or block.

> ### Caution
>
> **Interrupt blocks and programs can interrupt the execution of non-interrupt logic as well as other Timed and I/O-Triggered programs. Therefore, 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 blocks and programs from executing when shared data is being accessed.**

Interrupt Handling and Scheduling with Blocks

An Interrupt block has the highest priority of any user logic in the system and may be programmed to execute upon the receipt of a Timed or I/O Interrupt block. The execution of a block triggered from a timed or I/O interrupt preempts the execution of the normal PLC sweep activities. Execution of the normal PLC sweep activities is resumed after the Interrupt block completes. There can be a maximum of 64 I/O Interrupt blocks and 16 Timed Interrupts blocks.

Note

Timer function blocks do not accumulate time if used in a block that is executed as a result of a Timed or I/O Interrupt block.

Beginning with Release 6 of the PLC CPU, LD Interrupt blocks may make calls to other blocks. The application stack used during the execution of Interrupt blocks is different from the stack used by the LD program. Therefore, the nested call limit is different from the limit described for calls from the _MAIN block. The PLC will log an "Application Stack Overflow" fault and the PLC will transition to Stop/Fault mode if a call results in insufficient stack space to complete the call.

Note

Blocks which may execute as a result of a timed or I/O interrupt should not be called from the _MAIN block or other Non-Interrupt blocks because portions of the code executed by blocks are not re-entrant. In the example below INT1, INT2, BLOCK5, and PSB1 should not be called from _MAIN, BLOCK2, BLOCK3, or BLOCK4.
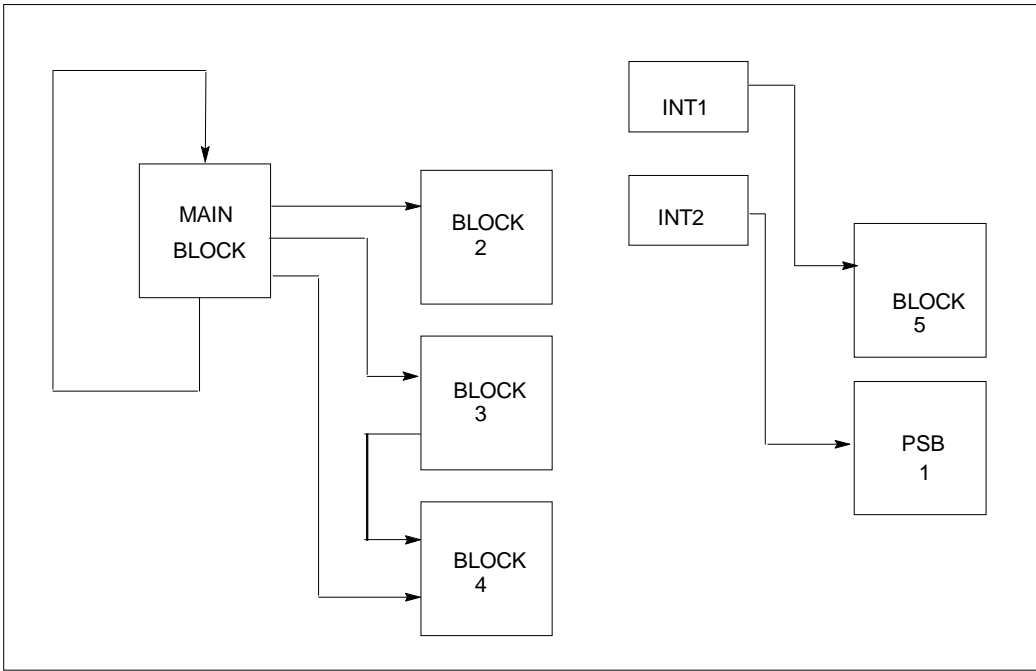
I/O Interrupt Blocks

A block may 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 module is configured in this manner, that input can serve as a trigger to cause the execution of an LD or External block.

To program an I/O Interrupt block, the block must first be declared in the programming software. It must then be associated with the interrupt through the use of an interrupt declaration.

The figure below shows two I/O interrupt declarations. The trigger ST_BUT calls LD block INT1 if the input from a stop button wired to input 1 transitions in the configured direction. The module can be configured to generate the interrupt on a rising edge or a falling edge of the input. The LD block INT2 is triggered by %AI00009.
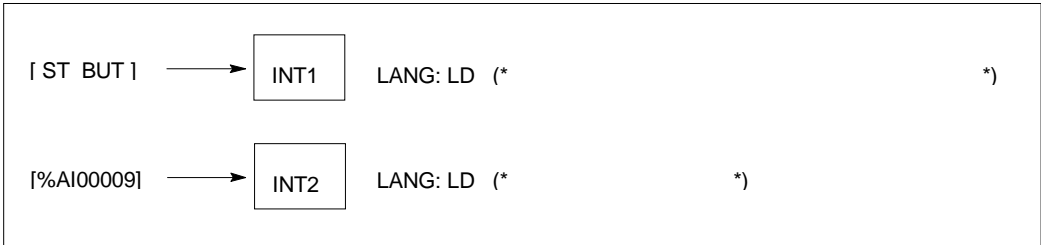
Figure 2-11. I/O Interrupt Block Declarations

Note

Parameterized subroutine blocks (PSBs) with zero parameters and External blocks (C blocks and C FBKs) with zero parameters may also be triggered by an interrupt input. (Zero parameter subroutine blocks are called SUBRs in Control software.) For these types of blocks, the local data (%L) is inherited from the _MAIN local data (%P), for example, %L0005 = %P0005.

Timed Interrupt Blocks

A block may be executed on a user-specified time interval with an initial delay (if specified) applied on a Stop-to-Run transition of the PLC. In Logicmaster, the time base options for Timed Interrupt blocks are 1.0 second, 0.10 second, 0.01 second, and 0.001 second. In Control software, the time base is 0.001 second (abbreviated *msec* within the Task Definitions dialog box).

To program a Timed Interrupt block, the block must first be declared. It must then be associated with a timed interrupt and given an interval and initial delay through the use of an interrupt declaration.

The first execution of a Timed Interrupt block will occur at ((DELAY * time base) + (INTVL * time base)) after the PLC is placed in Run mode. The figure below shows two timed interrupt declarations. The LD block BLK1 will be executed at times of 3 seconds, 5 seconds, 7 seconds, etc., after the PLC is placed in Run mode. The LD block BLK2 will be executed at two-second intervals, beginning two seconds after the PLC is placed in Run mode. The absence of a DELAY value for BLK2 indicates that there will not be an initial delay in the first execution of the block.

```
              _____
             |      |
             |TIMER |———————▶  |  CALL     BLK1 |  LANG: LD    (*                        *)
             |      |          |_____|
   CONST     |1.00s |
   00002 ———|INTVL |
             |      |
   CONST     |      |
   00001 ———|DELAY |
             |_____|


              _____
             |      |
             |TIMER |———————▶  |  CALL     BLK2 |  LANG: LD    (*                        *)
             |      |          |_____|
   CONST     |1.00s |
   00002 ———|INTVL |
             |      |
           _ |      |
             |DELAY |
             |_____|
```
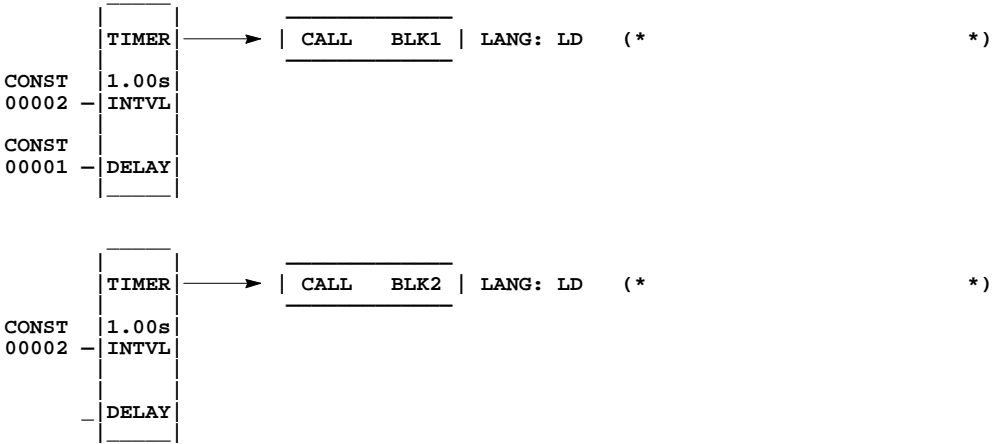
Figure 2-12. Timed Interrupt Block Declarations

Parameters:

Parameter	Description
INTVL	INTVL is a constant value which will be multiplied by the time base of the interrupt to establish the frequency of execution of the associated block.
DELAY	DELAY is an optional field for the timed interrupt. It is a constant value which will be multiplied by the interrupt time base to establish an additional delay for the first execution of the associated block.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
INTVL															•	
DELAY															•	•

• = Valid data type, or place where power may flow through the function.

Note

Parameterized Subroutine Blocks (PSBs) with zero parameters and External blocks (C blocks and C FBKs) with zero parameters may also be triggered by a timed interrupt. (Zero parameter subroutine blocks are called SUBRs in Control software.) For these types of blocks, the local data (%L) is inherited from the _MAIN local data (%P), for example, %L0005 = %P0005.

Interrupt Handling and Scheduling with User Programs

I/O-Triggered Programs

Beginning with Release 6 of the PLC CPU, one of the scheduling modes available for user programs is the activation of programs from an I/O interrupt. The I/O-Triggered scheduling mode allows a user program to be invoked, along with its corresponding input and output specification copy, when a configured I/O interrupt occurs. I/O-Triggered programs execute during any phase of the PLC sweep or only during the Logic Window, based on sweep mode. Refer to "User Program Execution" earlier in this section for more information on the scheduling and execution of I/O-Triggered programs.

To program an I/O-Triggered program, the program must first be declared in the programming software. The scheduling mode must then be set to I/O Interrupt Triggered.

Timed Programs

Beginning with Release 6 of the PLC CPU, one of the scheduling modes available for user programs is the activation of a program from a timed interrupt. The Timed scheduling mode allows a user program to be executed, along with its corresponding input and output specification copy, on a user-specified time interval with an initial delay (if specified) applied on a Stop-to-Run transition of the PLC. Timed programs execute during any phase of the PLC sweep. (Refer to the "User Program Execution" earlier in this section for more information on the execution of Timed programs.)

To program a Timed program, the program must first be declared. The scheduling mode must then be set to Timed Interrupt.

The time base options for Timed programs are specified in milliseconds. The first execution of a Timed program will occur at [(Initial Delay) + (Time Interval * time base)] milliseconds after the PLC is placed in Run mode.

Interrupt Blocks vs. Interrupt Programs

There are important differences to be aware of when choosing a program instead of a block to handle an interrupt. When a block is selected to handle a Timed or I/O Interrupt, the block will execute immediately upon receipt of the interrupt and run until it completes. Interrupt blocks can execute during any phase of the PLC sweep, regardless of the current PLC sweep mode. Pending Timed Interrupt blocks will execute before pending I/O Interrupt blocks, but once an Interrupt block (Timed or I/O) begins executing, it will run until it completes. If an interrupt occurs which attempts to execute a Timed or I/O Interrupt block which has not fully completed execution due to a previous interrupt, the interrupt will be queued and the block will be executed again after the Interrupt block completes execution. If an Interrupt block has already been queued in this manner once, any additional interrupts that occur for this block will be ignored.

Upon receipt of the Interrupt, Timed or I/O-Triggered programs are immediately scheduled to begin execution (including the copying of the input specification). However, the actual execution of the program occurs on a priority basis. Unlike Interrupt blocks, the execution of Timed or I/O-Triggered programs can be delayed or preempted by other Timed or I/O-Triggered programs of a higher priority as well as other Interrupt blocks. Additionally, if an interrupt occurs which attempts to schedule a Timed or I/O-Triggered program which has not fully completed execution due to a previous interrupt, a "Program not Readied" application fault will be logged in the PLC fault table and the interrupt will be ignored.

When the PLC is in Normal Sweep, Constant Sweep, or Constant Window mode, Interrupt programs can execute during *any* phase of the PLC sweep. When the PLC is in Microcycle Sweep mode, Interrupt programs are scheduled to execute in the Logic Window.

In summary, the primary differences between Interrupt blocks and Interrupt programs are as follows:

Interrupt Block

- Executed immediately upon receipt of interrupt

- Cannot be preempted by other logic once Interrupt block begins execution

- One additional interrupt for this block is "queued" if the block is still executing due to a previous interrupt

- Executes during any phase of the PLC sweep

Interrupt Program

- Scheduled to execute on a priority basis

- Can be preempted by higher priority Interrupt program or Interrupt block

- Additional interrupts for this program are ignored and a fault is logged if the program is still executing due to a previous interrupt

- Executes during any phase of the PLC sweep when PLC is in Normal Sweep, Constant Sweep, or Constant Window mode

- Executes in Logic Window when PLC is in Microcycle Sweep mode

Section 5: Run/Stop Operations

Modes of Operation

Four run/stop modes of operation are supported by the 90-70 PLC. You can change these modes in the following ways: the toggle switch, 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 PLC toggle switch, passwords, etc.

Run/Outputs Enabled	In this mode, the PLC runs user programs and continually scans inputs and updates physical outputs, including Genius and Field Control outputs. The Programmer and System Communications Windows are run in either Limited, Run-to-Completion, or Constant mode.
Run/Outputs Disabled	In this mode, the PLC 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 Programmer and System Communications Windows are run in either Limited, Run-to-Completion, or Constant mode.
Stop/IO Scan	In this mode the PLC does not run user programs, but the inputs and outputs are scanned. The Programmer and System Communications Windows are run in Run-to-Completion mode. The Background Window is limited to 10 ms.
Stop/No IO Scan	In this mode the PLC does not run user programs, and the inputs and outputs are not scanned. The Programmer and System Communications Windows are run in a Run-to-Completion mode. The Background Window is limited to 10 ms.

Note

You cannot store changes to %P and %L references 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.

Note

Stop/IOScan is not supported in Microcycle Sweep mode for all Release 6 CPUs.

Mode Transitions

Stop-to-Run Transition

Several operations are performed by the CPU on Stop-to-Run transition. These operations include the following:

- 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-to-Stop Transition

Wind-Down Period for Microcycle Sweep Mode

When the PLC is running in Microcycle Sweep mode (refer to section 4 for information about Microcycle Sweep mode), a wind-down or logic solution period may occur after the PLC is commanded to Stop mode. This wind-down period is equal to the amount of time that the currently executing program(s) take to complete their execution unless that amount of time exceeds 2.5 seconds. If the currently executing programs exceed 2.5 seconds in their attempt to complete their executions, a fault will be logged in the PLC fault table, and the CPU will complete its transition to Stop mode. During the wind-down period, no additional programs (including Interrupt programs and blocks) will be scheduled for execution. Input Scans, Output Scans, Communications Windows (Programmer and System), and the Background Window continue during the wind-down period.

Note

By definition, exceeding the CPU wind-down period means that not all programs completed execution prior to the PLC going to Stop mode. Furthermore, when the PLC is next commanded to Run mode, all programs will begin execution at their normal beginning point. Program(s) are **not** resumed at their "wind-down exceeded" execution point.

Section 6: Power-Up and Power-Down Sequences

Power-Up

System power-up consists of the following parts:

- Power-up self-test
- PLC 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. Series 90-70 PLC modules execute 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.

In the CPU, power-up will be either a quick power-up (a warm start) or a full power-up (a cold start), depending on whether the CPU is able to go to Run mode after powering up. If all the conditions are met for the CPU to go to Run mode—a valid program or configuration is present, the switch is in Run mode, and no fatal fault exists—then the CPU will perform a quick power-up. If any of the conditions are not met, a full power-up is performed.

A quick power-up will only perform the CPU processor and BCP (Boolean Coprocessor) tests, along with a minimal RAM test. The goal of a quick power-up is to get the CPU up and running as quickly as possible. The remaining tests, ROM CRC, exhaustive memory tests, and peripheral tests are only performed on a full power-up.

If a low battery indication is present, then a low battery fault is logged into the PLC fault table.

PLC Memory Validation

The next phase of system power-up is the validation of the PLC memory within the CPU. 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, then a checksum is calculated across the _MAIN ladder block. If no ladder diagram program exists, then a checksum is calculated across the smallest standalone C program.

When the system is sure that the application RAM is preserved, then a known area of the BCP (Boolean Coprocessor) bit cache area is checked to determine if the BCP bit cache data was preserved. If this test passes, then 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 this checksum does not compare 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 PLC 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 PLC.

System Configuration

After completing its own self-test, the CPU performs the system configuration. It first clears all of the system diagnostic bits in the BCP (Boolean Coprocessor) 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, checking for completion of the module's self-test.

The CPU reads information from each module, comparing it with user-provided rack/slot configuration information. Any differences between actual configuration and user-specified 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 a Model 70 input module, no further configuration is required. If the module is a Model 70 output module, the module is commanded to go to its default state. A Model 70 output module defaults to all outputs off at power-up and in failure mode, unless told otherwise. When the module is a Bus Transmitter Module (BTM), it is interrogated about what remote racks are present in the system. Based upon the BTM'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 user-specified 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.

For Model 70 input modules, the board may be set to interrupt when the signal(s) change state, and whether the interrupt will occur when the signal(s) transitions from high to low or low to high. For Model 70 output modules, their default state may be changed from Off to Hold Last State.

Power-Down Sequence

System power-down occurs when the power supply detects that incoming AC power has dropped for more than one power cycle. A signal line on the backplane is driven low to indicate the condition, which causes an interrupt to the CPU. From the time this signal occurs, a minimum of 5 milliseconds remain to complete power-down processing.

Retention of Data Memory Across Power Failure

Because application RAM and BCP memory are 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)

Section 7: Clocks and Timers

Clocks and timers provided by the Series 90-70 PLC include an elapsed time clock, a time-of-day clock, and software and hardware watchdog timers. Three types of timer function blocks include an on-delay timer, an off-delay timer, and a start-reset timer. Timed contacts cycle on and off (in square-wave form) every 0.01 second, 0.1 second, 1.0 second, and 1 minute.

Elapsed Time Clock

The elapsed time clock uses 100 microsecond "ticks" to track the time elapsed since the CPU powered on. The clock is not retentive across a power failure; it restarts on each power-up. Once per second the hardware interrupts the CPU to enable a seconds count to be updated. 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 function #16.

Time-of-Day Clock

The time of day in the Series 90-70 PLC is maintained by a hardware time-of-day clock. 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 time-of-day clock is battery-backed and maintains its present state across a power failure. However, unless the user initializes the clock, the values it contains are meaningless. The application program can read and set the time-of-day clock using Service Request function #7. The time-of-day clock can also be read and set from your programming software.

The time-of-day clock is designed to handle month-to-month and year-to-year transitions. It automatically compensates for leap years throuh year 2038.

Watchdog Timer

Software Watchdog Timer

A software watchdog timer in the Series 90-70 PLC is designed to detect "failure to complete sweep" conditions. The timer value for the software watchdog timer is set by using your 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 which prevents the PLC 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 and Microcycle Sweep modes, allowance for oversweep conditions should be considered when selecting the software watchdog timer value.

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 PLC fault table, and outputs go to their default state. If you are using serial or WSI communications (not Ethernet), the CPU will only communicate with the programmer; no other communications or operations are possible. To recover, power must be cycled on the rack 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.

Hardware Watchdog Timer

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

Section 8: System Security

The Series 90-70 PLC supports the following three types of system security:

1. Passwords/privilege levels

2. OEM protection

3. Write protect keyswitch

Passwords and Privilege Levels

Passwords are a configurable feature of the Series 90-70 PLC. Their use is optional and may be set up using your programming software. The purpose of passwords is to provide different levels of access privilege for the PLC when the programmer is in Online or Monitor mode. Passwords are not used if the programmer is in Offline mode. The use of passwords may restrict the following:

-   Changing I/O and PLC configuration data
-   Changing programs
-   Reading PLC data
-   Reading programs
-   Locking blocks

The default state is no password protection. There is one password for each privilege level in the PLC. Each password may be unique; however, the same password can be used for more than one level. Passwords are one to seven ASCII characters in length. Only the programmer may change passwords.

PLC password protection can be used to restrict access to selected PLC functions. After passwords have been set up, access to the PLC via any communications path is restricted unless the proper password has been entered. Once a password has successfully been accepted, access to the privilege level requested and below will be granted (for example, provide password for level 3 will allow access to functions at levels 0, 1, 2, and 3). If the PLC communications are suspended, protection level will automatically return to the lowest privilege level of the highest unprotected level or privilege level 2.

Table 2-18. Privilege Levels

Priv Level	Password	Access Description
4	Yes	Write to all 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.
3	Yes	Write to all configuration or logic when the CPU is in Stop mode, including word-for-word changes, the addition/deletion of program logic, and the overriding of discrete I/O.
2	Yes	Write to any data memory. This includes the toggle/force of reference values but does **not** include overriding discrete I/O. The PLC can be started or stopped. PLC and I/O fault tables can be cleared. **NOTE:** This is the default if no passwords are defined.
1	Yes	Read any PLC data, except for passwords. This includes reading fault tables, performing datagrams, verifying logic/config, load program and configuration, etc. from the PLC. NONE of this data may be changed. At this level, transition to RUN mode from the programmer is not allowed.
0	No	Read the current status of the PLC (including features supported by the PLC), read the name of the Resource (CP name prior to release 6.0), change privilege level, and log in as programmer. At this level, transition to RUN mode from the programmer is not allowed.

Note

The user must be aware that the RUN mode switch on the CPU overrides the 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.

Protection Level Request from Programmer

Upon connection to the CPU, the Programmer then requests the CPU to move to the highest non-protected level, thereby giving the programmer access to the highest non-protected level without having to specifically request any particular level.

A 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 PLCs password access table for the requested level, the privilege level change is denied and a fault is logged in the PLC fault table. The current privilege level is maintained, and no change will occur. 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 the user desires to prevent the use of password protection, passwords can be disabled using the programming software.

Note

To re-enable passwords after passwords have been disabled, the PLC must be power-cycled with the battery removed.

Password protection also prevents firmware upgrades to the FLASH memories used on the CPM 914, 915, 924, 925, and all CPX models. Prior to attempting a firmware upgrade in any of these modules, disable password protection, then re-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 PLC 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 and writes of the logic as well as writes to the configuration are prohibited. This allows a third-party OEM to create Control Programs for the PLC CPU and then set the OEM-locked mode which prevents the end-user from reading or modifying the program.

Note

OEM protection also prevents firmware upgrades to the FLASH memories used on the CPM 914, 915, 924, 925, and all CPX models. Prior to attempting a firmware upgrade in any of these modules, disable OEM protection, then enable it again after the upgrade.

Write Protect Keyswitch

The 90-70 CPU models CPU 780, 781, 782, 788, 789, 790, 914, 924, 915, 925; CGR 772 and 935; and all CPX models contain a memory write protect keyswitch. This keyswitch is located on the top of the faceplate, above the upper faceplate-to-rack clip. When in the protected position, the PLC program and configuration cannot be modified or deleted.

Note

The write protect keyswitch, when in the "write protected" position, also prevents firmware upgrades to the FLASH memories used on the CPM 790, 914, 915, 924, 925, and all CPX models. Prior to attempting a firmware upgrade in any of these modules, place the write protect keyswitch into the "write enabled" position.

Section 9: Series 90-70 PLC I/O System

The Series 90-70 PLC I/O system provides the interface between the Series 90-70 PLC and user-supplied devices and equipment. The I/O system supports the rack-type Model 70 I/O, the Genius I/O system, and the FIP I/O system. A Genius I/O Bus Controller (GBC) module provides the interface between the Series 90-70 PLC CPU and a Genius I/O bus. A FIP I/O Bus Controller (FBC) module provides the interface between the Series 90-70 PLC CPU and a FIP I/O bus. In addition to supporting these three I/O subsystems, the I/O system will also support Ethernet Interfaces and PCMs.

The I/O structure for the Series 90-70 PLC is shown in the following figure:
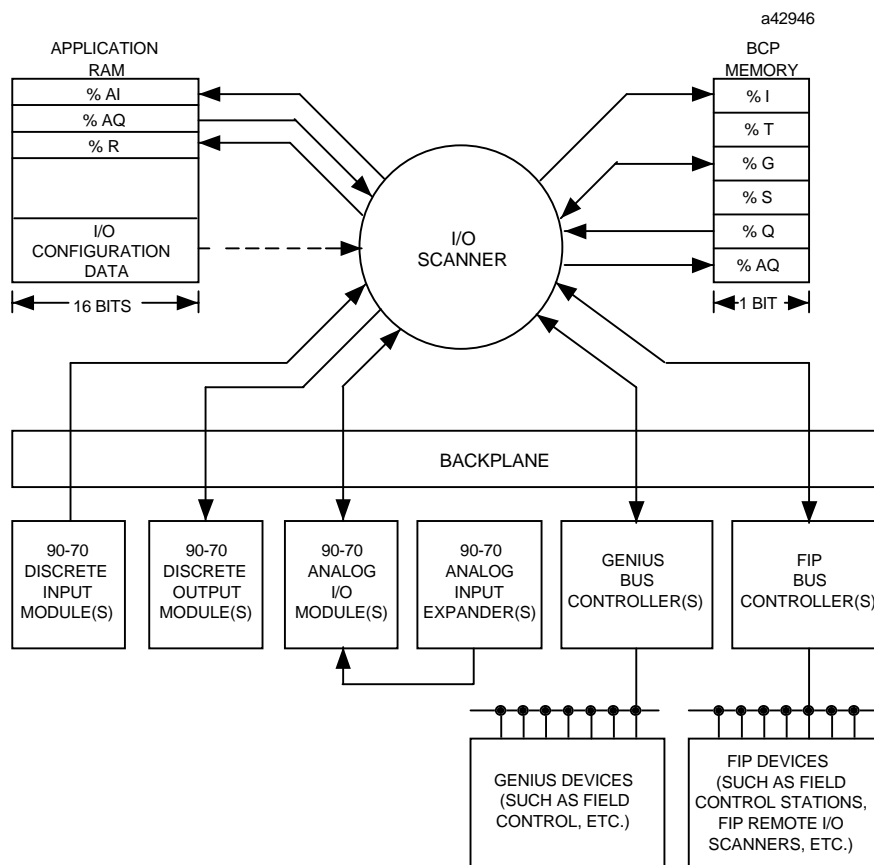
Figure 2-13.   Series 90-70 PLC I/O Structure

I/O Data Mapping

Discrete inputs and outputs are stored as bits in the CPU BCP Bit Cache memory. Analog I/O is stored in the application RAM allocated for that purpose. Analog data is always stored in the demultiplexed state, with each channel requiring one word (16 bits).

Default Conditions

The programming software provides the ability to specify that the first input may be an interrupt input and for the filter speed to be fast or slow; but upon power-up, Model 70 discrete input modules always default to the first input on the module not interrupting and the input filter being slow speed. If changed by the user, new defaults are applied when the board is configured by the CPU during the power-up process or whenever else the module may go through configuration.

Model 70 discrete output modules default to all outputs off. The configuration utility provides the ability to specify the default which will be applied when the CPU transitions from Run/Enabled to Run/Disabled or Stop mode. It also applies this default information when the system halts.

Genius I/O

Information relative to using Genius I/O in a Series 90-70 PLC system is presented in the following paragraphs. 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.

Genius I/O Bus Configuration

The Bus Controller used in the Series 90-70 PLC controls a single Genius I/O bus. Any type of Genius I/O block 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 Genius Bus Controller.

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 8 I/O 115 VAC Discrete Grouped Block (IC660CBD100) requires 8 bits in the %I memory and 8 bits in the %Q memory, regardless of how the block is configured.

The following four Genius I/O blocks are assigned to the analog memories:

- 6-Channel Analog Grouped Block
- 6-Channel Thermocouple Block
- 6-Channel RTD Block
- 4-Channel Strain Gauge/mV Analog Input Block

The Thermocouple and RTD blocks are also referred to as Low-Level Analog Input blocks.

Analog Grouped Block

The Analog Grouped block contains four analog input channels and two analog output channels. When a 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 Bus Controller 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 are input-only blocks. All have six channels.

Default Conditions

Genius I/O blocks have a number of default conditions that may be set using the Genius I/O Hand-Held Monitor. These defaults include the following:

- Report faults
- Range select
- Analog input and output scaling
- Input filter time
- Alarm input mode
- Output hold last state
- Output default

These defaults are stored in EEPROM in the block itself. The Series 90-70 PLC configuration utility supports the changing of only a small subset of these defaults. For more information, 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 Bus Controller to change any default condition on a specific block. However, this change will only be accepted by the block 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 *Genius Bus Controller User's Manual*, GFK-0398.

Genius Global Data Communications

The Series 90-70 PLC supports the sharing of data among multiple PLC 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 PLCs that have Genius I/O capability can send global data to a Series 90-70 PLC and can receive data from a Series 90-70 PLC. Your 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 90-70 PLC is in STOP/NOIO mode. However, if the 90-70 PLC is in STOP/IOSCAN mode, then Genius global data communications will continue to operate.

FIP I/O

Information relative to using FIP I/O in a Series 90-70 PLC system is presented in the following paragraphs. For specific information on FIP I/O types, configuration, and setup, refer to the *Series 90-70 FIP Bus Controller User's Manual*, GFK-1038.

FIP I/O Bus Configuration

The FIP Bus Controller used in the Series 90-70 PLC controls a single FIP I/O bus. Currently supported are the 90-30 FIP Remote I/O Scanner, FIP Bus Interface Unit (for Field Control), and generic FIP I/O module configurations. All of the FIP I/O interface modules (for example, the FIP Remote I/O Scanner) must provide input data to the 90-70 FIP Bus Controller so that the FIP Bus Controller has this same input data to provide to the 90-70 CPU during the next normal input scan. Similarly, when the 90-70 CPU performs the next output scan, the FIP Bus Controller accepts this output data and passes it on to the appropriate FIP I/O interface module to then update the local I/O.

90-30 FIP Remote I/O Scanner

The FIP Remote I/O Scanner provides the ability to use 90-30 I/O as a remote I/O node on a FIP I/O network. The FIP Remote I/O Scanner module provides the communications interface to the FIP I/O network (communications with the 90-70 FIP Bus Controller) and also provides the I/O scanning function for the local 90-30 I/O modules. (For more information on the 90-30 FIP Remote I/O Scanner, please refer to the *90-30 FIP Remote I/O Scanner User Manual,* GFK-1037)

FIP Bus Interface Unit (Field Control)

The FIP Bus Interface Unit provides the ability to use Field Control I/O as a remote I/O node on a FIP I/O network. The FIP Bus Interface Unit module provides the communication interface to the FIP I/O network (communications with the 90-70 FIP Bus Controller) and also provides the I/O

scanning function for the local Field Control modules. (For more information on FIP Field Control, please refer to the *FIP Bus Interface Unit User's Manual*, GFK-1175).

Generic FIP I/O

Generic FIP I/O allows for configuring FIP I/O other than the FIP Remote I/O Scanner and FIP Bus Interface Unit. This permits the 90-70 CPU and the 90-70 FIP Bus Controller to assign I/O reference addresses to the generic FIP I/O device. The configuration selection also permits the 90-70 FIP Bus Controller to recognize the generic FIP I/O module on the FIP I/O network.

FIP I/O Fault Data

In the I/O fault table, the rack, slot, FIP drop ID, remote rack, and remote slot number are given for faults occurring in an FIP Remote I/O Scanner or FIP Bus Interface Unit controlled remote I/O node. No fault information can be obtained from generic FIP I/O.

FIP I/O Data Mapping

FIP I/O discrete inputs and outputs are stored as bits in the CPU Bit Cache memory. FIP 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, an analog module with four inputs and two outputs consumes four words of Analog Input memory (%AI) and two words of Analog Output memory.

Default Conditions

FIP I/O devices have a number of default conditions which may be set using your programming software. The default conditions include the following:

- Range select
- Analog input scaling
- Analog output scaling
- Discrete output default Off (fixed)
- Analog output default Hold Last State (fixed)

Diagnostic Data Collection

Diagnostic data in a Series 90-70 PLC I/O system is obtained in one of the following two ways:

1. If an I/O module has an associated Bus Controller (Genius Bus Controller or FIP Bus Controller), then the Bus Controller provides the module's diagnostic data for the CPU.

2. If an I/O module is a Model 70 I/O module, then the CPU's I/O Scanner subsystem generates the diagnostic bits based on the data provided by the I/O module.

The diagnostic bits are derived from the diagnostic data sent from the I/O modules to their I/O controllers (Genius Bus Controller, FIP Bus Controller, or 90-70 CPU). Diagnostic bits always indicate the current fault status of the associated module. Bits are set when faults occur and are cleared when faults are cleared.

In general, diagnostic data is not maintained by the Series 90-70 PLC for foreign I/O (not GE Fanuc) modules. Any diagnostic information provided by those boards must be specifically accessed by the application program using the VME Read and VME Write function blocks.

Beginning with 90-70 CPU release 5.50, the 90-70 system supports third-party I/O modules when developed under license agreement with GE Fanuc. These boards are then configured as "3rd PartyVME" and the interface mode is "I/O Scan." Boards developed to conform to the I/O Scan interface can provide discrete and analog diagnostic information to the 90-70 CPU.

Discrete I/O Diagnostic Information

Diagnostic information is maintained by the Series 90-70 PLC for each discrete I/O point. Two memory blocks are allocated in application RAM for discrete diagnostic data. One is associated with %I memory and the other with %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 available that may be interrogated using two special contacts: a fault contact (-[FAULT]-) and a no-fault contact (-[NOFLT]-). The PLC only collects this fault data if enabled to do so through your 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 PLC CPU for each analog channel associated with Model 70 analog input modules, Model 70 analog output modules, Genius analog blocks, etc. Two memory blocks are allocated for analog diagnostic data. One is associated with %AI analog input memory and the other with %AQ analog output memory. One byte of diagnostic memory is allocated for 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 -[FAULT]- and -[NOFLT]- contacts. The process bits are referenced with the -[HIALR]- and -[LOALR]- contacts. The memory allocation for analog diagnostic data is one byte per word of analog input and analog output allocated by the user. When an analog fault contact is referenced in the application program, the PLC 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.

Chapter 3

Fault Explanation and Correction

This chapter is an aid to troubleshooting a Series 90-70 PLC system. It explains the fault descriptions, which appear in the PLC fault table, and the fault categories, which appear in the I/O fault table.

Each fault explanation in this chapter lists the fault description for the PLC fault table or the fault category for the I/O fault table. Find the fault description or fault category corresponding to the entry on the applicable fault table displayed on your programmer screen. Beneath it is a description of the cause of the fault along with instructions to correct the fault.

Chapter 3 contains the following sections:

Section	Title	Description	Page
1	System Handling of Faults (General)	Describes the PLC system faults (SY_FLT) and the I/O faults (IO_FLT). Describes configurable faults, changing the fault action, non-configurable faults, and locating fault references (rack, slot, bus, and FIP locating references).	3-2
2	Fault Handling	Describes the type of faults that may occur in the Series 90-70 PLC and how they are displayed in the fault tables. Descriptions of the PLC and I/O fault table displays are also included.	3-10
3	PLC Fault Table Explanations	Provides a fault description of each PLC fault and instructions to correct the fault.	3-16
4	I/O Fault Table Explanations	Provides a description of each I/O fault and instructions to correct the fault.	3-38

Section 1: System Handling of Faults (General)

The system fault references listed below can be used to identify the specific type of fault that has occurred.

System Fault Reference	Description
ANY_FLT	Any fault in the system.
SY_FLT	Any system fault in the Series 90 PLC.
IO_FLT	Any I/O fault.
SY_PRES	Indicates a new entry in the PLC fault table.
IO_PRES	Indicates a new entry in the I/O fault table.
HRD_FLT	Any hardware fault.
SFT_FLT	Any software fault.

On power-up, the system fault references are cleared. If a fault occurs, the on-transition state of the affected reference(s) is on the next sweep after the fault occurs. The system fault references remain on as long as the fault exists, until the PLC is cleared or until cleared from the program. The ANY_FLT fault is set when any other fault is set. The SY_PRES and IO_PRES faults are set when the PLC and I/O fault table contain entries.

System Fault References

When a system fault reference is set, additional fault references are also set. The following table lists these other types of faults. References marked with asterisks below are configurable system fault references.

Table 3-1. System Fault References

Fault Type	PLC System Fault (SY_FLT)	I/O Fault (IO_FLT)
Hardware Fault (HRD_FLT)	SBUS_ER System bus error HRD_CPU PLC CPU hardware fault. HRD_SIO Module hardware fault SBUS_FL System bus failure *	
Software Fault (SFT_FLT)	SFT_SIO Intelligent module software fault SFT_CPU PLC software fault * MAX_IOC Too many Bus Controllers * STOR_ER Programmer download failed *	SFT_IOC I/O Controller software fault
Other Faults	PB_SUM block checksum fault LOW_BAT Low battery signal OV_SWP Over constant sweep time SY_FULL PLC fault table full IO_FULL I/O fault table full APL_FLT Application program fault NO_PROG No application program at power-up * BAD_RAM Corrupted program memory * WIND_ER Incomplete window service * BAD_PWD Password access failure * NUL_CFG No configuration present * LOS_SIO Loss of option module. * ADD_RCK Addition of expansion rack * ADD_SIO Addition of option module * CFG_MM Configuration mismatch * LOS_RCK Loss of rack *	LOS_IOC Loss of I/O Controller LOS_IOM Loss of I/O module ADD_IOC Addition of I/O controller ADD_IOM Addition of I/O module IOC_FLT Bus or I/O Controller fault IOM_FLT I/O module fault

* Configurable system fault references.

Configurable Fault Actions

For some faults, the PLC must stop execution. For other faults, the appropriate response to a fault may depend on the nature of the application. All faults are initially assigned to one of these three actions:

Fault Action	Description
Fatal	These faults halt the system, set diagnostic variables, and are logged in a fault table.
Diagnostic	These faults do not halt the system. They do, however, set diagnostic variables and are logged in a fault table.
Informational	These faults are logged in a fault table, but cause no other action.

For some faults, called "non-configurable" faults, the fault action cannot be changed. Other faults, called "configurable" faults, can have their fault type changed to another fault action if such a change is suitable for the application. The following table lists configurable faults:

Table 3-2. Fault References for Configurable Faults

Fault (Default Action)	Description	May Also Be Set
SBUS_ER (diagnostic)	System bus error. (The BSERR* signal was generated on the VME system bus.)	HRD_FLT SY_PRES, SY_FLT Other references may also be set depending on the type of access when the BSERR* occurred.
SFT_IOC* (diagnostic)	Non-recoverable software error in a I/O Controller.	IO_FLT, IO_PRES SFT_FLT
LOS_RCK** (diagnostic)	Loss of rack (BRM failure, loss of power), or missing a configured rack.	SY_FLT, SY_PRES IO_FLT, IO_PRES
LOS_IOC*** (diagnostic)	Loss of I/O Controller channel, or missing a configured Bus Controller.	IO_FLT, IO_PRES
LOS_IOM (diagnostic)	Loss of I/O module (does not respond), or missing a configured I/O module.	IO_FLT, IO_PRES
LOS_SIO (diagnostic)	Loss of intelligent module (does not respond), or missing a configured module.	SY_FLT, SY_PRES
IOC_FLT (diagnostic)	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)	Wrong module type detected during power-up or Run mode. The PLC does not check the configuration parameters set up for individual modules such as Genius I/O blocks.	SY_FLT, SY_PRES

* The SFT-IOC software fault will have the same action as what you set for LOS_IOC.
** 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.
*** Even if the LOS-IOC fault is configured as Fatal, the PLC will not go to STOP/FAULT unless both Genius Bus Controllers 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.

Non-Configurable Faults

For non-configurable faults, the fault action cannot be changed.

The following table lists non-configurable faults.

Table 3-3. Non-Configurable Faults

Fault	Description	Result
SBUS_FL (fatal)	System bus failure. The PLC CPU was not able to access the VME bus. BUSGRT*NMI error.	Sets SY_FLT, HRD_FLT, and SY_PRES.
HRD_CPU (fatal)	PLC CPU hardware fault, such as failed memory device or failed serial port).	SY_FLT, SY_PRES HRD_FLT
HRD_SIO (diagnostic)	Non-fatal hardware fault on any module in the system, such as failure of a serial port on a PCM.	SY_FLT, SY_PRES HRD_FLT
SFT_SIO (diagnostic)	Non-recoverable software error in a PCM or LAN interface module.	SY_FLT, SY_PRES SFT_FLT
PB_SUM (fatal)	Program or block checksum failure during power-up or in Run mode.	SY_FLT, SY_PRES
LOW_BAT (diagnostic)	Low battery signal from CPU or another module in system.	SY_FLT, SY_PRES
OV_SWP (diagnostic)	Constant sweep time exceeded.	SY_FLT, SY_PRES
SY_FULL IO_FULL (diagnostic)	PLC fault table full (16 entries). I/O fault table full (32 entries).	SY_FLT, SY_PRES IO_FLT, IO_PRES
APL_FLT (diagnostic)	Application fault.	SY_FLT, SY_PRES
ADD_RCK** (diagnostic)	New rack added, or previously faulted rack has returned.	SY_FLT, SY_PRES
ADD_IOC (diagnostic)	Previously faulted I/O Controller is no longer faulted.	IO_FLT, IO_PRES
ADD_IOM (diagnostic)	Previously faulted I/O module is no longer faulted.	IO_FLT, IO_PRES
ADD_SIO (diagnostic)	New intelligent module is added, or previously faulted module no longer faulted.	SY_FLT, SY_PRES
IOM_FLT (diagnostic)	Point or channel on an I/O module; a partial failure of the module.	IO_FLT, IO_PRES

Table 3-3. Non-Configurable – Continued

Fault	Description	Result
NO_PROG (information)	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.	Does not set any references. 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)	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)	Window completion error. Servicing of Programmer or Logic Window was skipped. Occurs in Constant Sweep or Microcycle Sweep Mode.	Sets SY_FLT and SY_PRES.
BAD_PWD (information)	Change of privilege level request to a protection level was denied; bad password.	Sets SY_FLT and SY_PRES.
NUL_CFG (fatal)	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.
SFT_CPU (fatal)	CPU software fault. A non-recoverable error has been detected in the CPU. May be caused by Watchdog Timer expiring.	PLC immediately transitions to Error Sweep 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.
MAX_IOC (fatal)	The maximum number of bus controllers has been exceeded. The Series 90 PLC supports 32 bus controllers.	Sets SY_FLT, SY_PRES, and SFT_FLT.
STOR_ER (fatal)	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.

Fault Contacts

Fault (-[FAULT]-) and no-fault (-[NOFLT]-) contacts can be used to detect the presence of various faults in the system. These contacts can not be overridden. The following table shows the state of fault and no-fault contacts.

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

Fault Locating References (Rack, Slot, Bus, Module)

The Series 90-70 PLC supports reserved fault names for each rack, slot, bus, and module if the hardware is configured. By programming these names on the FAULT and NOFLT contact instructions, logic can be executed to locate faults associated with configured racks and modules.

Format of Fault References

Table 3-4. Fault Reference Names

Fault Reference Type	Reserved Name	Comment
Rack	RACK_0r	Where r is rack number 0 to 7.
Slot	SLOT_rs	Where r is rack number 0 to 7 and s is slot number 0 to 9.
Bus	BUS_rsb (Genius only)	Where r is rack number 0 to 7, s is slot number 0 to 9, and b is the Genius bus number 1 or 2.
Module	M_rsbmm (Genius only)	Where r is rack number 0 to 7, s is slot number 0 to 9, b is the Genius bus number 1 or 2, and mm is the Serial Bus Address (SBA) number 00 to 31.
FIP Module	F_rsmmm (FIP only)	Where r is rack number 0 to 7, s is slot number 0 to 9, and mmm is the FIP station address 000 to 255.

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.

These fault names do not correspond to %SA, %SB, %SC, or to any other reference type. Only the name is displayed. A reference table screen is not provided for the fault references.

The format of a rack fault name is RACK_0r, where r is the rack number 0 to 7. For example, RACK_01 shown in the example below represents rack 1.

The format of a slot fault name is SLOT_rs, where r is the rack number 0 to 7 and s is the slot number 0 to 9. For example, SLOT_15 shown in the example below represents rack 1, slot 5.

```
|
| RACK_01 SLOT_15                                                          %Q00002
|─[FAULT]─[NOFLT]───────────────────────────────────────────────────────────( )─
|
```

The format of a bus fault name is a BUS_rsb, where r is the rack number 0 to 7, s is the slot number 0 to 9, and b is the bus number 1 or 2. For example, BUS_241 represents rack 2, slot 4, bus 1.

The format of a module fault name is M_rsbmm, where r is the rack number 0 to 7, s is the slot number 0 to 9, b is the bus number 1 or 2, and mm is the module number 00 to 31. For example, M_26128 represents rack 2, slot 6, bus 1, module 28.

The format of a FIP module fault is F_rsmmm, where *r* is the rack number 0 to 7, *s* is the slot number 0 to 9, and *mmm* is the FIP station address.

Behavior of Fault 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.)

- For FIP faults, refer to the *Series 90-70 FIP Bus Controller User's Manual* (GFK-1038).

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 3-9) 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 faults references are set: RACK_05, SLOT_56, BUS_561, and M_56129. There will only be one entry in the fault table to describe the problem with the module. The fault table does not show entries pertaining to the rack, slot, and bus in this case. A fault in FIP device at rack 5, slot 6, module 29 results in the following fault references being set: RACK_05, SLOT_56, F_56029.

A module fault for a FIP Remote I/O Scanner device results in only one module fault type fault reference being set. For example, if FIP module with station ID 43 in 90-70 rack 3, slot 9 is an FIP Remote I/O Scanner, and within the FIP Remote I/O Scanner there is a faulted module in the FIP Remote I/O Scanner's rack 1, slot 5, then only fault reference F_39043 will be set.

Alarm Contacts

High (-[HIALR]-) and low (-[LOALR]-) alarm contacts are used to represent the state of the analog input module comparator function. However, the use of point faults must first be enabled in Hardware Configuration. For details about enabling Point Fault References, refer to the Logicmaster User's Manual (GFK-0263) or the online help in Hardware Configuration within CIMPLICITY Control.

The following example logic uses both high and low alarm contacts.

```
|
|%AI0001                                                                  %Q00003
|─[HIALR]+───────────────────────────────────────────────────────────────( )─
|        |
|%AI0002 |
|─[LOALR]+
|
```
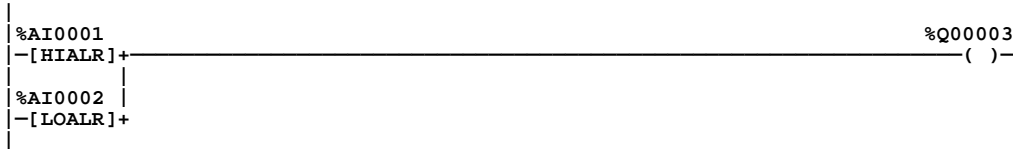
Note

HIALR and LOALR contacts will not create an entry in a fault table.

Point Faults

Point faults pertain to external I/O faults, although they will also be set due to the failure of associated higher-level internal hardware (for example, IOC failure or loss of a rack). In order to use point faults, they must be enabled in Hardware Configuration. For details about enabling Point Fault References, refer to the Logicmaster User's Manual (GFK-0263) or the online help in Hardware Configuration within CIMPLICITY Control.

When enabled, a Boolean reference for each discrete I/O point fault and a byte reference for each analog I/O channel level fault 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 above provide access to the point fault.

Section 2: Fault Handling

Faults occur in the Series 90-70 PLC system when certain failures or conditions happen which affect the operation and performance of the system. These conditions, such as the loss of an I/O module or rack, may affect the ability of the PLC to control a machine or process. These conditions may also have beneficial effects, such as when a new module comes online and is now available for use. Or these conditions may only act as an alert, such as a low battery signal which indicates that the battery protecting the memory needs to be changed.

For information on system status/fault references, refer to Chapter 2, section 3, "Program Organization."

Alarm Processor

The condition or failure itself is called a fault. When a fault is received and processed by the CPU, it is called an alarm. The software in the CPU which handles these conditions is the Alarm Processor. The interface to the user for the Alarm Processor is through the programming software. Any detected fault is recorded in a fault table and displayed on either the PLC fault table screen or the I/O fault table screen, as applicable.

Classes of Faults

The Series 90-70 PLC detects several classes of faults. These include internal failures, external failures, and operational failures.

Table 3-5. Classes of Faults

Fault Class	Examples
Internal Failures	Non-responding modules. Low battery condition. Memory checksum errors.
External I/O Failures	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 Reaction to Faults

Typically, hardware failures require that either the system be shut down or the failure be tolerated. I/O failures may be tolerated by the PLC system, but they may be intolerable by the application or the process being controlled. Operational failures are normally tolerated. Series 90-70 PLC faults have three attributes:

Table 3-6. Fault Attributes

Attribute	Description
Fault Table Affected	I/O fault table PLC fault table
Fault Action	Fatal Diagnostic Informational
Fault Response	Configurable Non-configurable

Fault Tables

The two fault tables—the PLC fault table and the I/O fault table—are provided in this chapter to make faults easier to find.

Fault Action

Fatal faults cause the fault to be recorded in the appropriate table, any diagnostic variables to be set, and the system to be stopped. Diagnostic faults are recorded in the appropriate table, and any diagnostic variables are set. Informational faults are only recorded in the appropriate table.

Table 3-7. Fault Actions

Fault Action	Response by CPU
Fatal	Log fault in fault table. Set fault references. Go to Stop mode.
Diagnostic	Log fault in fault table. Set fault references.
Informational	Log fault in fault table.

Your programming software provides the capability to change the fault action of certain faults. There are two possible classifications in the utility: fatal and non-fatal. These correspond to fatal and diagnostic fault action in the PLC. Only fatal faults cause the system to halt. Additionally, the informational fault action only logs faults in the fault table.

When a fault is detected by the CPU, it uses a default fault action for that fault. For those faults which may have their action changed by the programming software, the CPU uses the fault action specified by the software; this may be the default action or the action chosen by the user.

Fault Response

Fault response refers to the ability of a fault to have its fault action changed. Those faults that can have their fault action changed are called configurable faults. Those which cannot are called non-configurable faults. Non-configurable faults are either fatal or informational. Also, non-configurable faults do not cause application available references to be set and cannot have alarm blocks associated with the detection of the fault. Some non-configurable faults also have other effects associated with them. Generally, these effects control the changing of the CPU's execution mode (Stop, Run/Disabled, Run/Enabled). An example of such an effect is the disabling of I/O when a null system configuration is detected in the system.

PLC Fault Table

The PLC fault table displays PLC faults such as password violations, PLC/configuration mismatches, parity errors, and communications errors.

The programmer may be in any operating mode. However, if the programmer is in Offline mode, no faults are displayed. In Online or Monitor mode, PLC fault data is displayed. In Online mode, faults can be cleared (this may be password protected).

Field	Description
Top Fault Displayed	The index of the PLC fault currently at the top of the fault display is shown on the first line of this screen.
Total Faults	The total number of faults since the table was last cleared.
Table Last Cleared	The date and time faults were last cleared from the fault table. This information is maintained by the PLC.
Entries Overflowed	The number of entries lost because the fault table has overflowed since it was cleared. The PLC fault table can contain up to 40 faults (16 prior to Release 6—configurable in CIMPLICITY Control).
PLC Time/Date	The current date and time. This is also maintained by the PLC.

Note

The size of the PLC fault table is configurable (with a default of 16 and a maximum of 40—configurable in CIMPLICITY Control). Additional faults (over the configured limit) cause the table to overflow, and faults are lost. The system reference SY_FULL (%S0009) is set to indicate that the fault table is full.

User-Defined Faults

User-defined faults can be logged in the PLC fault table. When a user-defined fault occurs, it is logged in the appropriate fault table as "Application Msg (error_code):" and may be followed by a descriptive message up to 24 characters. All characters in the descriptive message can be defined by the user. 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).

Note

User-defined faults are created using Service Request 21 (refer to Chapter 4 in this manual).

I/O Fault Table

The I/O fault table displays I/O faults such as circuit faults, address conflicts, forced circuits, and I/O bus faults.

The programmer may be in any operating mode. However, if the programmer is in Offline mode, no faults are displayed. In Online or Monitor mode, PLC fault data is displayed. In Online mode, faults can be cleared (this feature may be password protected).

Field	Description
Top Fault Displayed	The index of the I/O fault currently at the top of the fault display is shown on the first line of this screen.
Total Faults	The total number of faults since the table was last cleared.
Fault Description	A more specific indication of the type of fault that is currently highlighted in the I/O fault table.
Table Last Cleared	The date and time faults were last cleared from the fault table. This information is maintained by the PLC.
Entries Overflowed	The number of entries lost because the fault table has overflowed since it was cleared. The I/O fault table can contain up to 32 faults.
PLC Time/Date	The current date and time. This is also maintained by the PLC.

Note

The size of the I/O fault table is configurable (with a default of 32 and a maximum of 40—configurable in CIMPLICITY Control). Additional faults (over the configured limit) cause the table to overflow, and faults are lost. The system reference IO_FULL (%S0010) is set to indicate that the fault table is full.

Accessing Additional Fault Information

The fault tables displayed by the programming software contain basic information regarding the fault. For additional information a pertaining to each fault, double-click the fault as it appears in the programming software to access the Details window.

The last entry, *Correction*, for each fault explanation in this chapter lists the action(s) to be taken to correct the fault. Note that the corrective action for some of the faults includes the statement:

**Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field
Service, giving them all the information contained in the fault entry.**

This second statement means that you must tell Field Service both the information readable directly from the fault table **and** the hexadecimal information. Field Service personnel will then give you further instructions for the appropriate action to be taken.

Section 3: PLC Fault Table Explanations

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 different fault conditions sharing the same fault description. The error code is the first two hexadecimal digits in the fifth group of numbers, as shown in the following example.

```
01  000000  01030100  0902  0200  000000000000
                                ┃ ┏━━━ Error Code (first two hex
                                ┗━━┛    digits in fifth group
```

Some faults can occur because random access memory on either the PLC CPU board or the expansion memory board has failed. These same faults may also occur because the system has been powered off and the battery voltage is (or was) too low to maintain memory. To avoid excessive duplication of instructions when corrupted memory may be a cause of the error, the correction simply states:

Perform the corrections for Corrupted Memory.

This means:

1. If the system has been powered off, replace the battery. Battery voltage may be insufficient to maintain memory contents.

2. Replace the expansion memory board. Integrated circuits on the memory board may be failing.

3. Replace the PLC CPU board. The integrated circuits on the PLC CPU board may be failing.

Note

For information about values for fault groups, refer to Appendix B.

Configurable Faults

Configurable faults can have their fault action (fatal or diagnostic) changed. The CPU uses the fault action specified by the software; this may be the default action or a fault action chosen by the user. In this section, the default fault action is listed for configurable faults.

Loss of or Missing Rack

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. The default fault action for this group is Diagnostic.

Error Code:	1
Name:	Rack Lost
Description:	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.
Error Code:	2
Name:	Rack Not Responding
Description:	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

The fault group Loss of or Missing Option Module occurs when a GEnet, PCM, BTM, or BRM fails to respond. The failure may occur at power-up if the module is missing or during operation if the module fails to respond. The default fault action for this group is Diagnostic.

Error Code:	3
Name:	Bus Transmitter Module Found in Expansion Rack
Description:	The PLC generates this error when a Bus Transmitter Module is found in an expansion rack.
Correction:	Power off the system and remove the BTM from the expansion rack.
Error Code:	16
Name:	Analog Expander Located to the Left of the Base Converter module.
Description:	An Analog Expander module has been placed in a rack to the left of its Base Converter module.
Correction:	Power off the system. Move the Analog Expander module to the right of the Base Converter module.
Error Code:	19
Name:	Lost Analog Expander module
Description:	Base Converter module has lost communications with the Analog Expander module.
Correction:	(1) Verify wiring linking Base Converter module with the Analog Expander module.
	(2) Replace the Analog Expander module.
	(3) If communication with all Analog Expanders is lost, replace the Base Converter module.
Error Code:	2C, 2D
Name:	Option Module Soft Reset Failed
Description:	PLC CPU unable to re-establish communications with option module after soft reset.
Correction:	(1) Try soft reset a second time.
	(2) Replace the option module.
	(3) 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.
	(4) Replace the cables.
	(5) Replace the BRM (Bus Receiver Module).
	(6) Replace the BTM (Bus Transmitter Module).
	(7) Report failure to GE Fanuc PLC Field Service.
Error Code:	3B
Name:	Loss of, or Missing Communications Driver
Description:	The PLC generates this error when VME communications fail between the PLC CPU and a third party VME module using the FULL MAIL configuration mode.
Correction:	(1) Update the configuration file with the correct communications parameters.
	(2) Replace the communications driver on the module.
	(3) Remove the module from the configuration file.
	(4) Replace the module.

Error Code:	3C
Name:	Module in Firmware Update Mode
Description:	The PLC generates this error when it finds a module in Firmware Update mode. Modules in this mode will not communicate with the PLC 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.
Error Code:	41
Name:	Unable to Establish VME Communications
Description:	The PLC generates this error when it finds a module in Standalone mode. A module in Standalone mode will appear to be operating correctly, but it will not communicate with the PLC CPU.
Correction:	(1) Reset the module with the push-button.
	(2) Power-cycle the entire system.
	(3) Power-cycle the rack containing the module.
Error Code:	FF
Name:	Option Module Communications Failed
Description:	PLC CPU generates this error when communication to the option module has failed.
Correction:	(1) Check the bus for abnormal activity.
	(2) Replace the intelligent option module to which the request was directed.
	(3) Check the parallel programmer cable for proper attachment.
Error Code:	4B
Name:	CFG_486_NOT_COMPAT
Description:	The currently installed CPX PLC firmware is not compatible with the ESCM firmare (Embedded Serial Communications Module firmware—the firmware that controls Serial Ports 1 and 2). Byte 0 = Minimum Required CPX major firmware revision; Byte 1 = Minimum Required CPX minor firmware revision; Byte 2 = Currently Installed major firmware revision; Byte 3 = Currently Installed minor firmware revision
Correction:	The CPX PLC firmware needs to be upgraded to be compatible with the ESCM firmware.
Error Code:	4C
Name:	CFG_ESCM_NOT_COMPAT
Description:	The currently installed ESCM firmware (Embedded Serial Communications Module firmware—the firmware that controls Serial Ports 1 and 2) is not compatible with the PLC firmware currently installed. Byte 0 = Minimum Required ESCM major firmware revision; Byte 1 = Minimum Required ESCM minor firmware revision; Byte 2 = Currently Installed major firmware revision; Byte 3 = Currently Installed minor firmware revision
Correction:	The ESCM firmware needs to be upgraded to be compatible with the CPX PLC firmware.
Error Code:	51
Name:	ESCM_RESET_REQ
Description:	The ESCM (Embedded Serial Communications Module) requested a RESET, which is currently not supported. Most probable cause is that the ESCM is being boot loaded.
Correction:	Cycle power when the ESCM completes the update.

Error Code:	All Others
Name:	Module Failure During Configuration
Description:	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

The fault group Addition of Extra Rack occurs when a configured expansion rack with which the PLC CPU could not communicate comes online or is powered on, or an unconfigured rack is found. The default fault action for this group is Diagnostic.

Error Code:	1
Name:	Extra Rack
Correction:	(1) Check rack jumper behind power supply for correct setting.
	(2) Update the configuration file to include the expansion rack.
	(3) Remove the expansion rack from the hardware configuration.
	<u>Note:</u> No correction necessary if rack was just powered on.

Reset of, Addition of, or Extra Option Module

The fault group Reset of, Addition of, or Extra Option Module occurs when an option module (PCM, BTM, etc.) comes online, is reset, or a module is found in the rack but none is specified in the configuration. The default fault action for this group is Diagnostic.

Error Code:	1
Name:	Extra Option Module
Correction:	(1) Update the configuration file to include the module.
	(2) Remove the module from the system.
Error Code:	2
Name:	Module Restart Complete
Description:	Restart of module is complete.
Correction:	None
Error Code:	3
Name:	LAN Interface Restart Complete, Running Utility
Description:	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).

System Configuration Mismatch

The fault group Configuration Mismatch occurs when the module occupying a slot is different from that specified in the configuration file. The default fault action is Fatal. When the I/O Scanner 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.

Error Code:	2
Name:	Genius I/O Block Number Mismatch
Description:	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.
Error Code:	4
Name:	I/O Type Mismatch
Description:	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.
Error Code:	7
Name:	Daughter Board Mismatch
Description:	The PLC generates this error when the configuration file indicates one size memory daughter (expansion) board should be on the PLC CPU and a different size is actually present.
Correction:	(1) Replace the module.
	(2) Replace the daughter board with the size indicated in the configuration file.
	(3) Update the configuration file to agree with the size of the daughter board actually installed on the PLC CPU.
Error Code:	8
Name:	Analog Expander Mismatch
Description:	The PLC 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.
Error Code:	9
Name:	Genius I/O Block Size Mismatch
Description:	The PLC generates this error when block configuration size does not match the configured size.
Correction:	Reconfigure the block.

Error Code:	A
Name:	Unsupported Feature
Description:	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.
Error Code:	B
Name:	Revision A of BTM not in Right-most Slot
Description:	The BTM (Revision A version) is not the right-most module in the rack.
Correction:	(1) Move the BTM to the right of all other modules in the rack.
	(2) Upgrade the BTM to a newer version (Revision B or higher).
Error Code:	E
Name:	LAN Duplicate MAC Address
Description:	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.
Error Code:	F
Name:	LAN Duplicate MAC Address Resolved
Description:	Previous duplicate MAC address has been resolved. The module is back on the network. This is an informational message.
Correction:	None required.
Error Code:	10
Name:	LAN MAC Address Mismatch
Description:	MAC address programmed by softswitch utility does not match configuration stored from software.
Correction:	Change MAC address on softswitch utility or in software.
Error Code:	11
Name:	LAN Softswitch/Modem mismatch
Description:	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.
Error Code:	17
Name:	Invalid Memory Reference
Description:	Memory references in the logic program exceed what is available.
Correction:	Update the configuration file and store it to the PLC.
Error Code:	1E
Name:	Reference Length Mismatch
Description:	The PLC generates this error when the I/O reference lengths specified in the configuration for this module do not match the actual data sizes reported by the board.
Correction:	Update the configuration file with the correct reference lengths.

Error Code:	1F
Name:	Invalid Configuration Parameters
Description:	The PLC generates this error when it determines that critical values in the module's configuration are unacceptable.
Correction:	Update the configuration file with the correct values.
Error Code:	20
Name:	New Configuration Requires Reset
Description:	The PLC generates this error when it determines that a store of configuration attempted to change critical configuration values for the specified module. The new configuration will not take effect until the module is reset.
Correction:	(1) Power-cycle the entire system.
	(2) Power-cycle the rack containing the module.
Error Code:	27
Name:	Unresolved or Disabled Interrupt Reference
Description:	The PLC 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.
Error Code:	1D
Name:	Incompatible Scheduling Mode
Description:	A program with a scheduling mode that is incompatible with the sweep mode has been stored. Logged on a Stop-to-Run transition.
Correction:	(1) Change the sweep mode and try again.
	(2) Change the scheduling mode or delete the offending program(s) from the program declaration screen.
Error Code:	24
Name:	I/O Specification Mismatch
Description:	The I/O specification of a program does not match the specification given in the program.
Correction:	Correct the mismatch between the I/O specification by changing the I/O specification declaration or the corresponding macro declaration in the C program source file.
Error Code:	25
Name:	Controller Reference Out of Range
Description:	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.
Error Code:	26
Name:	Bad Program Specification
Description:	The I/O specification of a program is corrupted.
Correction:	Contact GE Fanuc Field Service.
Error Code:	All Others
Name:	Module and Configuration Do not Match
Description:	The PLC 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

The fault group System Bus Error occurs when the PLC CPU receives a non-configurable interrupt bus error from the bus system. The default fault action is Diagnostic.

Error Code:	4
Name:	Unrecognized VME Interrupt Source
Description:	The PLC generates this error when a module generates an interrupt not expected by the CPU (unconfigured or unrecognized).
Correction:	(1) Ensure that all modules configured for interrupts have corresponding interrupt declarations in the program logic.
	(2) Ensure that no third-party VME module is generating interrupts on the IRQ6 and IRQ7 lines.
Error Code:	All Others
Name:	System Bus Error
Description:	The PLC generates this fault when it has detected an error signal on the VME backplane, such as a parity error.
Correction:	(1) Ensure that all expansion rack cables are properly connected and seated.
	(2) Take action to minimize system noise.

PLC CPU Hardware Failure

The fault group PLC CPU Hardware occurs when the PLC CPU detects a hardware failure, such as a RAM failure or a communications port failure. When the failure is a RAM failure, the address of the failure is stored in the first four bytes of the Fault Extra Data field.

When a PLC CPU Hardware failure occurs, the PLC OK LED will flash on and off to indicate that the failure was not serious enough to prevent programmer communications to retrieve the fault information. The default fault action for this group is Fatal.

Error Code:	6Eh
Name:	Time-of-Day Clock not Battery-Backed
Description:	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.
Error Code:	All Others
Correction:	Replace the module.

Module Hardware Failure

The fault group Module Hardware Failure occurs when the PLC CPU detects a non-fatal hardware failure on any module in the system, for example, a serial port failure on a PCM. The default fault action for this group is Diagnostic.

Error Code:	1A0
Name:	Missing 12 Volt Power Supply
Description:	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.
Error Code:	1C2 - 1C6
Name:	LAN Interface Hardware Failure
Description:	Refer to the LAN Interface manual, GFK-0868 or GFK-0869 (previously GFK-0533), for a description of these errors.
Error Code:	All Others
Name:	Module Hardware Failure
Description:	A module hardware failure has been detected.
Correction:	Replace the affected module.

Option Module Software Failure

The fault group Option Module Software Failure occurs when a non-recoverable software failure occurs on a PCM. It is also generated when 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 default fault action for this group is Fatal.

Error Code:	1
Name:	Unsupported Board Type
Description:	The PLC 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 PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.
Error Code:	2, 3
Name:	COMMREQ Frequency Too High
Description:	COMMREQs are being sent to a module faster than it can process them.
Correction:	Change the PLC program to send COMMREQs to the affected module at a slower rate or monitor the completion status of each COMMREQ before sending the next.
Error Code:	4
Name:	More Than One BTM in a Rack
Description:	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.
Error Code:	191, 195
Name:	LAN Interface Software Failure
Description:	Refer to the LAN Interface manual, GFK-0868 or GFK-0869 (previously GFK-0533), for a description of these errors.
Error Code:	All Others
Name:	Option Module Software Failure
Description:	Software failure detected on an option module.
Correction:	(1) Reload software into the indicated module.
	(2) Replace the module.

Program or Block Checksum Failure

The fault group Program or Block Checksum Failure occurs when the PLC 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 PLC fault table record contains the name of the program or block in which the error occurred. The default fault action for this group is Fatal.

Error Code:	All
Name:	Program or Block Checksum Failure
Description:	The PLC generates this error when a program or block is corrupted.
Correction:	(1) Clear PLC memory and retry the store.
	(2) Examine C application for errors.
	(3) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Low Battery Signal

The fault group Low Battery Signal occurs when the PLC CPU detects a low battery on the PLC CPU board, the PLC CPU memory daughter board, or a module such as the PCM reports a low battery condition. The default fault action for this group is Diagnostic.

Error Code:	0
Name:	Failed Battery Signal
Description:	The CPU module (or other module having a battery) battery is dead.
Correction:	Replace the battery. Do not remove power from the rack until replacement is complete.
Error Code:	1
Name:	Low Battery Signal
Description:	A battery on the CPU or other module has a low signal.
Correction:	Replace the battery. Do not remove power from the rack until replacement is complete.

Constant Sweep or Microcycle Time Exceeded

The fault group Constant Sweep or Microcycle Time Exceeded occurs when the PLC CPU operates in Constant Sweep or Microcycle 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. The default fault action for this group is Diagnostic.

Error Code:	0 Constant Sweep 1 Microcycle
Correction:	If Constant Sweep: (1) Increase constant sweep time. (2) Remove logic from application program. If Microcycle: (1) Increase the time. (2) Modify execution intervals to give programs more time to execute.

PLC System Fault Table Full

The fault group PLC System Fault Table Full occurs when the PLC Fault Table reaches its maximum configured limit (see page 3-13). The default fault action for this group is Diagnostic.

Error Code:	0
Correction:	Clear the PLC fault table.

I/O Fault Table Full

The fault group I/O Fault Table Full occurs when the I/O Fault Table reaches its maximum configured limit (see page 3-14). To avoid loss of additional faults, clear the earliest entry from the table. The default fault action for this group is Diagnostic.

Error Code:	0
Correction:	Clear the I/O fault table.

Application Fault

The fault group Application Fault occurs when the PLC CPU detects a fault in the user program. The default fault action for this group is Diagnostic.

Error Code:	1
Name:	Indirect Address Out of Range
Description:	The PLC generates this error when one of the parameters to a function block is an indirect reference (that is, the parameter is an address within that memory type which contains the parameter value) and the contents of the indirect reference are out of range for the memory type. For example, consider a system with 500 %R registers defined. This fault would be generated if the parameter address were %R00100, and the contents of %R00100 were greater than 500 or zero.
	The Fault Extra Data field contains in the first two bytes the offset address of where the call was made, the segment selector and offset (reference) in the next four bytes, and the name of the program or block in which the function call resides in the next eight bytes.
Correction:	(1) Correct the indirect reference.
	(2) Increase the number of registers available, if possible.
Error Code:	2
Name:	Software Watchdog Timer Expired
Description:	The PLC generates this error when the watchdog timer expires. The PLC CPU stops executing the user program and enters Stop mode. The only recovery is to cycle power to the PLC CPU. Examples causing timer expiration: 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.
Error Code:	5
Name:	COMMREQ WAIT Mode Not Supported
Description:	The module receiving the COMMREQ does not support WAIT mode COMMREQs.
Correction:	Use NOWAIT mode COMMREQs.
Error Code:	6
Name:	COMMREQ Bad Task ID
Description:	The task selected by the COMMREQ does not exist on the option module.
Correction:	Correct the task ID.
Error Code:	7
Name:	Application Stack Overflow
Description:	Block call depth has exceeded the PLC capability.
Correction:	Increase the program's stack size or adjust application program to reduce nesting.
Error Code:	8 through D
Name:	LAN Interface Application Faults
Description:	Refer to the LAN Interface manual, GFK-0868 or GFK-0869 (previously GFK-0533), for a description of these errors.

Error Code:	0E
Name:	External Block Run-Time Error
Description:	A run-time error occurred during execution of an external block.
Correction:	Based on the fault information, correct the specific problem in the external block.
Error Code:	0F
Name:	SORT Interrupt Error
Description:	A SORT function executed in a timed or I/O interrupt at the same time a SORT function was executing in another block.
Correction:	Do not use the SORT function in both Interrupt and Non-Interrupt blocks.
Error Code:	11
Name:	Standalone Run-Time Error
Description:	A run-time error occurred during execution of a Standalone program.
Correction:	Based on the fault information, correct the specific problem in the standalone program.
Error Code:	1C
Name:	Program Exceeded Wind Down
Description:	A program failed to complete execution within the wind-down period (currently 2.5 seconds) after the PLC was commanded to stop.
Correction:	A program has gone into an infinite loop or is taking too long to execute. Correct the coding error or modify the program.
Error Code:	1D
Name:	Program Not Readied
Description:	A program scheduled to be readied has not completed its previous execution. The base cycle time is too small (Periodic programs), or the interrupt rate is too high (I/O-Triggered or Timed programs).
Correction:	(1) Increase the base cycle time or decrease the interrupt rate.
	(2) Increase the execution interval time to allow the program to finish execution.

Non-Configurable Faults

The fault action of Non-Configurable Faults cannot be changed. Fatal faults cause the PLC to enter a form of Stop mode at the end of the sweep in which the error occurred. Diagnostic faults are logged and corresponding fault contacts are set. Informational faults are simply logged in the PLC fault table.

System Bus Failure

The fault group System Bus Failure occurs when the PLC CPU software receives the non-configurable interrupt bus failure from the bus system. The default fault action for this group is Fatal.

Error Code:	1
Name:	Bus Grant Failure
Description:	The PLC operating software generates this error when the PLC CPU is unable to obtain control of the VME bus when required.
Correction:	(1) Ensure that any non-GE Fanuc boards which can become bus masters are relinquishing control of the VME bus when requested to do so by the PLC CPU. (2) Replace the PLC CPU module.

No User Program on Power-Up

The fault group No User Program on Power-Up occurs when the PLC CPU powers up with its memory preserved but no user program exists in the PLC. The PLC CPU detects the absence of a user program on power-up; the controller stays in Stop mode, performing the Stop mode sweep until a valid program is downloaded. The default fault action for this group is Informational.

Correction:	Download an application program before attempting to go to Run mode.

Corrupted User Program on Power-Up

The fault group Corrupted User Program on Power-Up occurs when the PLC CPU detects corrupted user RAM. The PLC CPU will remain in Stop mode until a valid user program and configuration file are downloaded. The default fault action for this group is Fatal.

Error Code:	1
Name:	Corrupted User RAM on Power-Up
Description:	The PLC generates this error when it detects corrupted user RAM on power-up.
Correction:	(0) Cycle power without battery. (2) Examine any C applications for errors. (3) Replace the battery on the PLC CPU. (4) Replace the expansion memory board on the PLC CPU. (5) Replace the PLC CPU.
Error Code:	2
Name:	Illegal Boolean OpCode Detected
Description:	The PLC generates this error when it detects a bad instruction in the user program.
Correction:	(1) Restore the user program and references, if any. (2) Examine any C applications for errors. (3) Replace the expansion memory board on the PLC CPU. (4) Replace the PLC CPU.
Error Code:	6
Name:	Corrupted Remote I/O Scanner EEPROM
Description:	The configuration in the Remote I/O Scanner EEPROM was found to be corrupted at power-up.
Correction:	Restore the Remote I/O Scanner configuration.

Window Completion Failure

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. The default fault action for this group is Informational.

Error Code:	0
Name:	Window Completion Failure
Description:	The PLC generates this error when the PLC 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.
Error Code:	1
Name:	Logic Window Skipped
Description:	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

The fault group Password Actual Failure occurs when the PLC CPU receives a request to change to a new privilege level and the password included with the request is not valid for that level. The default fault action for this group is Informational.

Error Code:	0
Correction:	Retry the request with the correct password.

Null System Configuration for Run Mode

The fault group Null System Configuration for Run Mode occurs when the PLC 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. The effect of this fault is to perform the function of a Suspend I/O. The default fault action for this group is Informational.

Error Code:	0
Correction:	Download a configuration file.

PLC CPU System Software Failure

Faults in the fault group PLC CPU System Software Failure are generated by the operating software of the Series 90-70 PLC CPU. They occur at many different points of system operation. When a fatal fault occurs, the PLC CPU immediately transitions into a special Error Sweep mode. The only activity permitted when the PLC is in this mode is communications with the programmer. The only method of clearing this condition is to cycle power on the PLC. The default fault action for this group is Fatal.

Error Code:	14, 27
Name:	Corrupted PLC Program Memory
Description:	The PLC generates these errors when certain PLC operating software problems occur. These should *not* occur in a production system.
Correction:	(1) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.
	(2) Perform the corrections for corrupted memory.
Error Code:	52
Name:	Backplane Communications Failed
Description:	The PLC generates this error when it attempts to comply with a request that requires backplane communications and receives a rejected mail response.
Correction:	(1) Check the bus for abnormal activity.
	(2) Replace the intelligent option module to which the request was directed.
	(3) Check parallel programmer cable for proper attachment.
Error Code:	5A
Name:	User Shut Down Requested
Description:	The PLC generates this informational alarm when SVCREQ #13 (User Shut Down) executes in the application program.
Correction:	None required. Information-only alarm.
Error Code:	7B
Name:	Remote I/O Scanner Communications Heartbeat Failure
Description:	Refer to the *Series 90-70 Remote I/O Scanner User's Manual*, GFK-0579, for a description of this error.
Correction:	None required. Information-only alarm.
Error Code:	94
Name:	Units Contain Mismatched Firmware, Update Recommended
Description:	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.
Error Code:	All Others
Name:	PLC CPU Internal System Error
Description:	An internal system error has occurred that should **not** occur in a production system.
Correction:	Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Too Many Bus Controllers

The fault group Too Many Bus Controllers occurs when the I/O Scanner portion of the PLC operating software detects that more than the maximum number (32) of bus controllers has been defined. The PLC CPU itself is a bus controller for the Model 70 I/O present in the system. The default fault action for this group is Fatal.

Note

Genius bus controllers which are configured for redundant and non-redundant blocks count as two bus controllers.

Correction:	(1)	Determine which modules are bus controllers and remove the extra ones.
	(2)	Delete a bus controller from the configuration file and store the file to the PLC CPU.
	(3)	If bus controllers have been moved from one slot in the rack to a different slot and this error did not occur before the move, cycle power on the rack. **No module should be inserted with power applied to rack.**
	(4)	Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Communications Failure During Store

The fault group Communications Failure During Store 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. If communications with the programming device performing the store is interrupted or any other failure occurs which terminates the store, this fault is logged. 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; the user must specifically clear the condition. The default fault action for this group is Fatal.

Error Code:	0
Correction:	Clear the fault and retry the download of the program or configuration file.

Run Mode Store Failure

Error Code:	1
Description:	Communications was lost, 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.
Error Code:	2
Description:	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.
Error Code:	3
Description:	Power was lost in the middle of a Run Mode Store.
Correction:	Delete and restore the program. This error is fatal.

Section 4: I/O Fault Table Explanations

The I/O fault table reports data about faults in three classifications:

- 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. For additional information pertaining to each fault, double-click the fault to access its Details window.

The following table describes the information provided with each fault category.

Table 3-8. Fault Category Descriptions

Fault Category	Fault Type	Fault Description	Fault Specific Data
Circuit Fault	Discrete Fault	Loss of User Side Power	Circuit Configuration *
		Short Circuit in User Wiring	Circuit Configuration *
		Sustained Overcurrent	Circuit Configuration *
		Low or No Current Flow	Circuit Configuration *
		Switch Temperature Too High	Circuit Configuration *
		Switch Failure	Circuit Configuration *
		Point Fault	Circuit Configuration *
		Output Fuse Blown	Circuit Configuration *
	Analog Fault	Input Channel Low Alarm	Circuit Configuration *
		Input Channel High Alarm	Circuit Configuration *
		Input Channel Under Range	Circuit Configuration *
		Input Channel Over Range	Circuit Configuration *
		Input Channel Open Wire	Circuit Configuration *
		Output Channel Under Range	Circuit Configuration *
		Output Channel Over Range	Circuit Configuration *
		Invalid Data	Circuit Configuration *
		Expansion Channel Not Responding	Circuit Configuration *
	Low-Level Analog Fault	Input Channel Low Alarm	Circuit Configuration *
		Input Channel High Alarm	Circuit Configuration *
		Input Channel Under Range	Circuit Configuration *
		Input Channel Over Range	Circuit Configuration *
		Input Channel Open Wire	Circuit Configuration *
		Wiring Error	Circuit Configuration *
		Internal Fault	Circuit Configuration *
		Input Channel Shorted	Circuit Configuration *
		Invalid Data	Circuit Configuration *
	GENA Fault	GENA Circuit Fault	GENA Fault Byte 2
	Remote I/O Scanner Fault	Remote I/O Scanner Circuit Fault	Byte 1: Circuit Type Byte 2: I/O Type
Loss of IOC			Timeout Unexpected State Unexpected Mail Status VME Bus Error
Addition of IOC			
Loss of I/O Module			
Addition of I/O Module			
Extra I/O Module			
Loss of Block	Fault Not Specified	Communications Lost	
Addition of Block			
Extra Block			
I/O Bus Fault	Bus Fault Bus Outputs Disabled		
Global Memory Fault			Subnet Group Number Global Variable Name

* Refer to table on next page.

Table 3-8. Fault Category Descriptions - Continued

Fault Category	Fault Type	Fault Description	Fault Specific Data
Module Fault	Headend Fault	EPROM or NVRAM Failure Calibration Memory Failure Shared Ram Failure Configuration MisMatch Watchdog Timeout Output Fuse Blown	
IOC Software Fault			
IOC Hardware Failure			
Forced Circuit			Block Configuration * Discrete/Analog Indication*
Unforced Circuit			Block Configuration * Discrete/Analog Indication*

*Refer to table below.

Three types of fault specific data occur in more than one fault category; they are block configuration, circuit configuration, and analog/discrete indication. The codings are shown in the following table.

Value	Description
	Circuit Configuration
1	Circuit is an input.
2	Circuit is an output.
3	Circuit is an output with feedback.
	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).
	Discrete/Analog Indication
1	Block is a discrete block.
2	Block is an analog block.

Circuit Fault

Circuit Fault has four fault types. Three of the four fault types have fault descriptions. Fault specific data is available for all faults. Circuit faults apply specifically to Genius I/O modules. The default fault action is Diagnostic. The following table describes the circuit fault category.

Table 3-9. Circuit Fault Category Description

Fault Category	Fault Type	Fault Description	Fault Specific Data
Circuit Fault	Discrete Fault	Loss of User Side Power	Circuit Configuration
		Short Circuit in User Wiring	Circuit Configuration
		Sustained Overcurrent	Circuit Configuration
		Low or No Current Flow	Circuit Configuration
		Switch Temperature Too High	Circuit Configuration
		Switch Failure	Circuit Configuration
		Point Fault	Circuit Configuration
		Output Fuse Blown	Circuit Configuration
	Analog Fault	Input Channel Low Alarm	Circuit Configuration
		Input Channel High Alarm	Circuit Configuration
		Input Channel Under Range	Circuit Configuration
		Input Channel Over Range	Circuit Configuration
		Input Channel Open Wire	Circuit Configuration
		Output Channel Under Range	Circuit Configuration
		Output Channel Over Range	Circuit Configuration
		Invalid Data	Circuit Configuration
		Expansion Channel Not Responding	Circuit Configuration
	Low-Level Analog Fault	Input Channel Low Alarm	Circuit Configuration
		Input Channel High Alarm	Circuit Configuration
		Input Channel Under Range	Circuit Configuration
		Input Channel Over Range	Circuit Configuration
		Input Channel Open Wire	Circuit Configuration
		Wiring Error	Circuit Configuration
		Internal Fault	Circuit Configuration
		Input Channel Shorted	Circuit Configuration
		Invalid Data	Circuit Configuration
	Remote Fault	Remote I/O Scanner Fault	

Discrete Fault

Discrete Fault has eight fault descriptions. More than one condition may be present in a particular reporting of the fault.

Name:	Loss of User Side Power
Description:	The Genius Bus Controller 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.
Name:	Short Circuit in User Wiring
Description:	The Genius Bus Controller 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.
Name:	Sustained Overcurrent
Description:	The Genius Bus Controller 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.
Name:	Low or No Current Flow
Description:	The Genius Bus Controller generates this error when there is very low or no current flow in the user circuit.
Correction:	Fix the cause of the condition.
Name:	Switch Temperature Too High
Description:	The Genius Bus Controller (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.
Name:	Switch Failure
Description:	The Genius Bus Controller (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.
Name:	Point Fault
Description:	The PLC 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.
Name:	Output Fuse Blown
Description:	The PLC 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, and replace the fuse. (2) Replace the block.

Analog Fault

Analog Fault has nine fault descriptions. More than one condition may be present in a particular reporting of the fault.

Name:	Input Channel Low Alarm
Description:	The Genius Bus Controller generates this error when the Genius Analog module reports a low alarm on an input channel.
Correction:	Correct the condition causing the low alarm.
Name:	Input Channel High Alarm
Description:	The Genius Bus Controller generates this error when the Genius Analog module reports a high alarm on an input channel.
Correction:	Correct the condition causing the high alarm.
Name:	Input Channel Under Range
Description:	The Genius Bus Controller generates this error when the Genuis Analog module reports an under-range condition on an input channel.
Correction:	Correct the problem causing the condition.
Name:	Input Channel Over Range
Description:	The Genius Bus Controller generates this error when the Genuis Analog module reports an over-range condition on an input channel.
Correction:	Correct the problem causing the condition.
Name:	Input Channel Open Wire
Description:	The Genius Bus Controller generates this error when the Genuis Analog module detects an open wire condition on an input channel.
Correction:	Correct the problem causing the condition.
Name:	Output Channel Under Range
Description:	The Genius Bus Controller generates this error when the Genius Analog module reports an under-range condition on an output channel.
Correction:	Correct the problem causing the condition.
Name:	Output Channel Over Range
Description:	The Genius Bus Controller generates this error when the Genius Analog module reports an over-range condition on an output channel.
Correction:	Correct the problem causing the condition.
Name:	Invalid Data
Description:	The Genius Bus Controller generates this error when it detects invalid data from a Genuis Analog input block.
Correction:	Correct the problem causing the condition.
Name:	Expansion Channel Not Responding
Description:	The PLC 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.

Low-Level Analog Fault

Low-Level Analog Fault has nine fault descriptions. More than one condition may be present in a particular reporting of the fault.

Name:	Input Channel Low Alarm
Description:	The Genius Bus Controller generates this error when the Genuis Analog module reports a low alarm on an input channel.
Correction:	Correct the condition causing the low alarm.
Name:	Input Channel High Alarm
Description:	The Genius Bus Controller generates this error when the Genuis Analog module reports a high alarm on an input channel.
Correction:	Correct the condition causing the high alarm.
Name:	Input Channel Under Range
Description:	The Genius Bus Controller generates this error when the Genuis Analog module reports an under-range condition on an input channel.
Correction:	Correct the problem causing the condition.
Name:	Input Channel Over Range
Description:	The Genius Bus Controller generates this error when the Genuis Analog module reports an over-range condition on an input channel.
Correction:	Correct the problem causing the condition.
Name:	Input Channel Open Wire
Description:	The Genius Bus Controller generates this error when the Genuis Analog module detects an open wire condition on an input channel.
Correction:	Correct the problem causing the condition.
Name:	Wiring Error
Description:	The Genius Bus Controller generates this error when the Genuis Analog module detects an improper RTD connections or thermocouple reverse junction fault.
Correction:	Correct the problem causing the condition.
Name:	Internal Fault
Description:	The Genius Bus Controller generates this error when the Genuis 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.
Name:	Input Channel Shorted
Description:	The Genius Bus Controller 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.
Name:	Invalid Data
Description:	The Genius Bus Controller generates this error when it detects invalid data from a Genuis 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 specific data.

Description:	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 IOC (I/O Controller)

The fault category Loss of IOC has no fault types or fault descriptions associated with it. The default fault action is Fatal.

Name:	Loss of or Missing IOC
Description:	The PLC generates this error when it cannot communicate with an I/O Controller and an entry for the IOC exists in the configuration file.
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 PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Addition of IOC (I/O Controller)

The fault category Addition of I/O Module has no fault types or fault descriptions associated with it. The default fault action for this category is Diagnostic.

Name:	Addition of IOC
Description:	The PLC generates this error when an IOC which 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.
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.

Loss of I/O Module

The fault category Loss of I/O Module applies to Model 70 I/O discrete and analog modules. There are no fault types or fault descriptions associated with this category. The default fault action is Diagnostic.

Name:	Loss of I/O Module
Description:	The PLC generates this error when it detects that a Model 70 I/O module is no longer responding to commands from the PLC CPU, or when the configuration file indicates an I/O module is to occupy a slot and no module exists in the slot.
Correction:	(1) Replace the module.
	(2) Correct the configuration file.
	(3) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Addition of I/O Module

The fault category Addition of I/O Module applies to Model 70 discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category. The default fault action is Diagnostic.

Name:	Addition of I/O Module
Description:	The PLC generates this error when an I/O module which had been faulted returns to operation.
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.

Extra I/O Module

The fault category Extra I/O Module applies only to Model 70 I/O modules. There are no fault types or fault descriptions associated with this category. The default fault action is Diagnostic.

Name:	Extra I/O Module
Description:	The PLC generates this error when it detects a Model 70 I/O module in a slot which 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.

Loss of Block

The fault category Loss of Block applies to Genius blocks. There are no fault types or fault descriptions associated with this category. The default fault action is Diagnostic.

Name:	Loss of Block
Description:	The PLC generates this error when it receives a Loss of Block fault from a Genius Bus Controller but the reason for the loss is unspecified.
Correction:	(1) Verify power and wiring to the block.
	(2) Replace the block.
Name:	Loss of Block - A/D Communications Fault
Description:	The Genius I/O operating software generates this error when it detects a loss of communications with a Genius I/O block.
Correction:	(1) Verify power and serial bus wiring to the block.
	(2) Replace the block.

Addition of Block

The fault category Addition of Block applies only to Genius blocks. There are no fault types or fault descriptions associated with this category. The default fault action is Diagnostic.

Name:	Addition of Block
Description:	The Genius operating software generates this error when it detects that a Genius block which stopped communicating with the controller starts communicating again.
Correction:	Informational only. None required.

Extra Block

The fault category Extra Block applies only to Genius I/O blocks. There are no fault types or fault descriptions associated with this category. The default fault action is Diagnostic.

Name:	Extra Block
Description:	The PLC operating software generates this error when it detects a Genius I/O block on the bus at a serial bus address which the configuration file should 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.

I/O Bus Fault

The fault category I/O Bus Faults has two fault types associated with it. The default fault action is Diagnostic.

Name:	Bus Fault
Description:	The Genius Bus Controller (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 is 10, meaning 10 errors in a 10 second period)
Correction:	(1) Determine the reason for the bus failure and correct it.
	(2) Replace the Genius Bus Controller.
	(3) The default of 10 (mentioned above) can be set higher if needed, but the bus should be examined electrically—use an oscilloscope for waveform check.
Name:	Bus Outputs Disabled
Description:	The Genius Bus Controller operating software generates this error when it times out waiting for the PLC CPU to perform an I/O scan.
Correction:	(1) Replace the PLC CPU.
	(2) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.
Name:	SBA Conflict
Description:	The Genius Bus Controller 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

The fault category Module Fault has one fault type, one headend fault, and eight fault descriptions. No fault specific data is present. The default fault action for this category is Diagnostic.

Name:	Configuration Memory Failure
Description:	The Genius Bus Controller generates this error when it detects a failure in a Genius block's EEPROM or NVRAM.
Correction:	Replace the Genius block's electronics module.
Name:	Calibration Memory Failure
Description:	The Genius Bus Controller generates this error when it detects a failure in a Genius block's calibration memory.
Correction:	Replace the Genius block's electronics module.
Name:	Shared RAM Fault
Description:	The Genius Bus Controller generates this error when it detects an error in a Genius block's shared RAM.
Correction:	Replace the Genius block's electronics module.
Name:	Watchdog Timeout
Description:	The PLC generates this error when it detects that a Model 70 input module watchdog timer has expired.
Correction:	Replace the Model 70 input module.
Name:	Output Fuse Blown
Description:	The PLC generates this error when it detects a blown fuse on a Model 70 output module.
Correction:	(1) Determine and repair the cause of the fuse blowing, and replace the fuse. (2) Replace the module.
Name:	Module Fault
Description:	An internal failure has been detected in a module.
Correction:	Replace the affected module.

IOC (I/O Controller) Software Fault

The fault category IOC Software Fault applies to any type of I/O Controller. There are no fault types or fault descriptions associated with it. The default fault action is Fatal.

Name:	Datagram Queue Full, Read/Write Queue Full
Description:	Too many datagrams or read/write requests have been sent to the Genius Bus Controller.
Correction:	Adjust the system to reduce the request rate to the Genius Bus Controller.
Name:	Response Lost
Description:	The Genius Bus Controller is unable to respond to a received datagram or read/write request.
Correction:	Adjust the system to reduce the request rate to the Genius Bus Controller.

IOC (I/O Controller) Hardware Failure

The fault category IOC Hardware Fault has no fault types or fault descriptions. The default fault action is Diagnostic.

Description:	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 Genius Bus Controller 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 Genius Bus Controller.
	(4) Selectively remove each block from the bus until the offending block is isolated then replace it.

Forced and Unforced Circuit

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. The default fault action is Informational.

Fault Specific Data contains data as shown below.

Byte Number	Description
1	Block Configuration
2	Analog/Discrete Information

Block Switch

The Fault Category Block Switch has no fault types or fault descriptions. The default fault action is Diagnostic.

Name:	Block Switch
Description:	The PLC 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 needs to be repaired.
	(a) Verify the bus wiring.
	(b) Replace the I/O controller.
	(c) Replace the Bus Switching Module (BSM).

Relay Functions

Programming consists of creating an application program for a PLC. Chapters 4 through 12 describe the programming instructions that can be used to create ladder logic programs for the Series 90-70 programmable controller.

If the Logicmaster 90 programming software is not yet installed, please refer to the *Logicmaster 90-70 Programming Software User's Manual*, GFK-0263, for instructions. The user's manual explains how to create, transfer, edit, and print programs.

Configuration is the process of assigning logical addresses, as well as other characteristics, to the hardware modules in the system. It may be done either before or after programming, using the configuration software; however, it is recommended that configuration be done first. If that has not been done, you should refer to the *Logicmaster 90-70 Programming Software User's Manual*, GFK-0263, to decide whether it is best to begin programming at this time.

This chapter explains the use of contacts, coils, and links in ladder logic rungs.

Using Contacts

A contact is used to monitor the state of a reference. Whether the contact passes power flow depends on the state or status of the reference being monitored and on the contact type. A reference is ON if its state is 1; it is OFF if its state is 0.

<p align="center">Table 4-1. Types of Contacts</p>

Type of Contact	Display	Contact Passes Power to Right:
Normally-open contact	–| |–	When reference is ON.
Normally-closed contact	–|/|–	When reference is OFF.
Positive transition contact	–|↑|–	If reference goes ON.
Negative transition contact	–|↓|–	If reference goes OFF.
Fault contact	–[FAULT]–	If reference has point fault.
No fault contact	–[NOFLT]–	If reference has no point fault.
High alarm contact	–HIALR]–	If reference exceeds high alarm.
Low alarm contact	–LOALR]–	If reference exceeds low alarm.
Continuation contact	<+>——	If the preceding continuation coil is set ON.

Using Coils

Coils are used to control discrete references. 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, an internal reference should be used for that coil or a continuation coil/contact combination may be used.

Coils are always located at the rightmost position of a line of logic. A rung may contain up to eight coils.

The type of coil used will depend on the type of program action desired. The states of retentive coils are saved when power is cycled or when the PLC goes from STOP to RUN mode. The states of non-retentive coils are set to zero when power is cycled or the PLC goes from **STOP** to **RUN** mode.

Table 4-2. Types of Coils

Type of Coil	Display	Power to Coil	Result
Coil (normally open)	–( )–	ON	Set reference ON.
		OFF	Set reference OFF.
Negated	–(/)–	ON	Set reference OFF.
		OFF	Set reference ON.
Retentive	–(M)–	ON	Set reference ON, retentive.
		OFF	Set reference OFF, retentive.
Negated Retentive	–(/M)–	ON	Set reference OFF, retentive.
		OFF	Set reference ON, retentive.
Positive Transition	–(↑)–	OFF→ON	If reference is OFF, set it ON for one sweep.
Negative Transition	–(↓)–	ON→OFF	If reference is OFF, set it ON for one sweep.
SET	–(S)–	ON	Set reference ON until reset OFF by -(R)-.
		OFF	Do not change the coil state.
RESET	–(R)–	ON	Set reference OFF until set ON by -(S)-.
		OFF	Do not change the coil state.
Retentive SET	–(SM)–	ON	Set reference ON until reset OFF by -(RM)-, retentive.
		OFF	Do not change the coil state.
Retentive RESET	–(RM)–	ON	Set reference OFF until set ON by -(SM)-, retentive.
		OFF	Do not change the coil state.
Continuation coil	—<+>	ON	Set next continuation contact ON.
		OFF	Set next continuation contact OFF.

Note

For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Normally Open Contact   –| |–

A normally open contact acts as a switch that passes power flow if the associated reference is ON (1).

Normally Closed Contact   –|\|–

A normally closed contact acts as a switch that passes power flow if the associated reference is OFF (0).

Example:

The following example shows a rung with 10 elements having nicknames from E1 to E10. Coil E10 is ON when reference E1, E2, E5, E6, and E9 are ON and references E3, E4, E7, and E8 are OFF.

```
|
|   E1      E2      E3      E4      E5      E6      E7      E8      E9      E10
—| |———| |———|/|———|/|———| |———| |———|/|———|/|———| |———(  )–
|
```
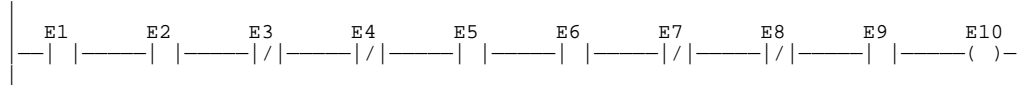
Positive Transition Contact   –|↑|–

A positive transition contact passes power flow if the associated reference transitions from OFF (0) to ON (1). The transition is determined from the write of an OFF (0) to the next write of an ON (1) value. These writes may occur multiple times during a PLC sweep, resulting in the transition bit being set for only a portion of the sweep; or they may occur several PLC sweeps apart, resulting in the transition bit being set for more than one sweep. Do not use the positive transition contact for references used with transition coils (also called one-shots) or SET and RESET coils.

Negative Transition Contact   –|↓|–

A negative transition contact passes power flow if the associated reference transitions from ON (1) to OFF (0). The transition is determined from the write of an ON (1) to the next write of an OFF (0) value. These writes may occur multiple times during a PLC sweep, resulting in the transition bit being set for only a portion of the sweep; or they may occur several PLC sweeps apart, resulting in the transition bit being set for more than one sweep. Do not use the negative transition contact for references used with transition coils (also called one-shots) or SET and RESET coils.

Additional Information on Transition Contacts

The transition bit for a reference point is affected when that point is written to. It is set when the point transitions from OFF to ON or from ON to OFF. It is reset when the state after the write is the same as the state before the write, i.e., ON to ON or OFF to OFF. The source of the write is immaterial; it can be an output coil, a function block output, the input scan, an input interrupt, a PCM SYSWRITE, a data change from the program, or Genius Datagram. When the point is written, the transition bit is immediately affected. Transition bits are not changed by the scan itself; only a write to the reference point. A write must be made to a reference in order to clear the transition bit, or it will appear to be "stuck." Nothing is done automatically per sweep to clear transition bits, except for configured input points, where a transition is cleared when the input data is read and the input point is in the same state as when read the previous sweep.

Overrides do not protect transition bits. If a write occurs to an overridden point, the transition bit is cleared. For example, the transition bit of an overridden input point is cleared when the input is scanned.

Example 1:

The following example shows the use of positive and negative transition contacts. Coil E2 is on for one logic sweep when element E1 transitions from OFF to ON. Coil E4 is ON for one sweep when element E3 transitions from ON to OFF.

```
 |
 |    E1                                                              E2
 |—( ↑ )—————————————————————————————————————————————————————————( )—
 |    E3                                                              E4
 |—( ↓ )—————————————————————————————————————————————————————————( )—
 |
```
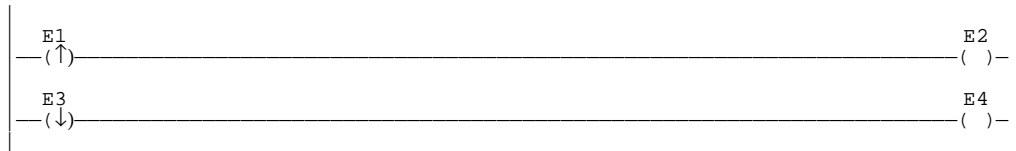
Example 2:

In the following example, bit %M00017 is set by a BSET function and then cleared by a BCLR function. The positive transition contact X1 activates the BSET, and the negative transition X2 activates the BCLR.

The positive transition associated with bit %M00017 will be on until %M00017 is reset by the BCLR 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 BSET function.

```
|         _____
|  X1    |     |                                                    E2
|--|↑|---|  BIT_|-------------------------------------------------( )-
|        | SET_ |
|        | WORD |
| %M00017-|IN    |
|        | LEN  |
|        |00001 |
|  CONST-|BIT   |
|  0001  |_____|
|
|         _____
|  X2    |     |                                                    E4
|--|↓|---|  BIT_|-------------------------------------------------( )-
|        | CLR_ |
|        | WORD |
| %M00017-|IN    |
|        | LEN  |
|        |00001 |
|  CONST-|BIT   |
|  0001  |_____|
|
```
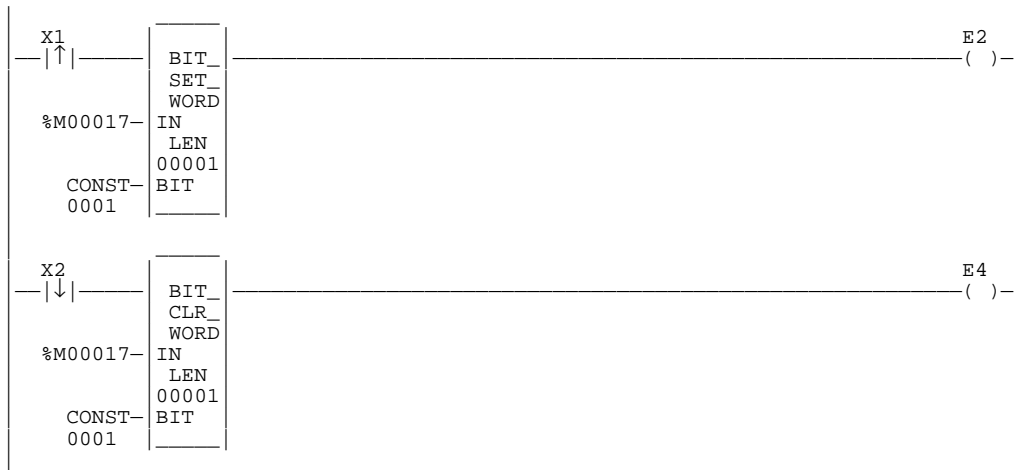
Fault Contact   –[FAULT]–

Fault contacts are used to detect faults in discrete or analog machine references, or to locate faults (rack, slot, bus). The use of I/O fault contacts must be enabled during CPU configuration if I/O point fault reporting is desired. (Refer to chapter 2, section 3, "Program Organization and User Data," for more information on point faults.)

A fault contact passes power flow if its associated variable or location has a point fault.

Note

Use the machine reference address (%I, %Q, %AI, %AQ) with the FAULT/NOFLT contacts to guarantee correct indication of module status. (When using the rack, slot, bus, module reference with a FAULT/NOFLT contact, the fault indication of a given module is cleared when the associated fault is cleared from the fault table.)

No Fault Contact   –[NOFLT]–

No fault contacts are also used to detect faults in discrete or analog machine references. The use of I/O No Fault contacts must also be enabled if I/O point fault reporting   is desired. (Refer to chapter 2, section 3, "Program Organization and User Data," for more information on point faults.)

A no fault contact passes power flow if its associated variable or location does not have a point fault.

Note

Use the machine reference address (%I, %Q, %AI, %AQ) with the FAULT/NOFLT contacts to guarantee correct indication of module status. (When using the rack, slot, bus, module reference with a FAULT/NOFLT contact, the fault indication of a given module is cleared when the associated fault is cleared from the fault table.)

High Alarm Contact   –[HIALR]–

The high alarm contact 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. (From the configuration software menu, select "Memory Allocation and Point Fault Enable". Change the setting for point fault reference from **DISABLED** to **ENABLED**).

A high alarm contact passes power flow if the high alarm bit associated with the analog reference is ON.

Low Alarm Contact   –[LOALR]–

The low alarm contact is used to detect a low alarm associated with an analog reference. Use of this contact must be enabled during CPU configuration, as described above for the high alarm contact.

A low alarm contact passes power flow if the low alarm bit associated with the analog reference is ON.

Coil   –( )–

A coil sets a discrete reference ON while it receives power flow. It is non-retentive; therefore, it cannot be used with system status references (%SA, %SB, %SC, or %G).

Example:

In the following example, coil E3 is ON when reference E1 is ON and reference E2 is OFF.

```
|   E1      E2                                                                E3
|──| |─────| / |──────────────────────────────────────────────────────────( )─
```
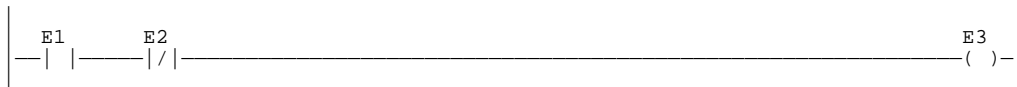
Negated Coil   –(/)–

A negated coil sets a discrete reference ON when it does not receive power flow. It is not retentive; therefore, it cannot be used with system status references (%SA, %SB, %SC, or %G).

Example:

In the following example, negated coil E2 is ON when reference E1 is OFF.

```
|   E1                                                                        E2
|──| |──────────────────────────────────────────────────────────────────( / )─
```

Retentive Coil   –(M)–

Like a normally open coil, the retentive coil sets a discrete reference ON while it receives power flow. The state of the retentive coil is retained across power failure. Therefore, it cannot be used with references from strictly non-retentive memory (%T).

Negated Retentive Coil   –(/M)–

The negated retentive coil sets a discrete reference ON when it does not receive power flow. The state of the negated retentive coil is retained across power failure. Therefore, it cannot be used with references from strictly non-retentive memory (%T).

Positive Transition Coil   –(↑)–

If the reference associated with a positive transition coil is OFF, when the coil receives power flow it is set to ON until the next time the coil is executed. (If the rung containing the coil is skipped on subsequent sweeps, it will remain ON.) This coil can be used as a one-shot.

Do not write from external devices (e.g., PCM) to references used on positive transition coils since it will destroy the one-shot nature of these coils.

NOTE

Do not use transition contacts on positive transition coils since the coil uses the transition bit to store the power flow value into the coil.

Transitional coils can be used with references from either retentive or non-retentive memory (%Q, %M, %T, %G, %SA, %SB, or %SC). ↓

Negative Transition Coil   –(↓)–

If the reference associated with this coil is OFF, when the coil _**stops**_ receiving power flow, the reference is set to ON until the next time the coil is executed.

NOTE

Do not write from external devices to references used on negative transition coils since it will destroy the one-shot nature of these coils.

Do not use transition contacts on negative transition coils since the coil uses the transition bit to store the power flow value into the coil.

Transitional coils can be used with references from either retentive or non-retentive memory (%Q, %M, %T, %G, %SA, %SB, or %SC).

Example:

In the following example, when reference E1 goes from OFF to ON, coils E2 and E3 receive power flow, turning E2 ON for one logic sweep. When E1 goes from ON to OFF, power flow is removed from E2 and E3, turning coil E3 ON for one sweep.

```
    E1                                                              E2
 —| |————————————————————————————————————————————————————————————(↑)–
    E1                                                              E3
 —| |————————————————————————————————————————————————————————————(↓)–
```
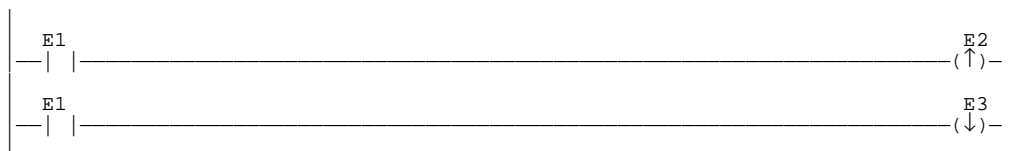
SET Coil   –(S)–

SET and RESET are non-retentive coils that can be used to keep ("latch") the state of a reference (e.g., E1) either ON or OFF. When a SET coil receives power flow, its reference stays ON (whether or not the coil itself receives power flow) until it is reset by power flow to a RESET coil of the same reference (e.g., E1 in the example below).

Note

SET coils write an undefined result to the transition bit for the given reference. ***Do not use transition contacts on references used on SET coils.***

RESET Coil   –(R)–

The RESET coil sets a discrete reference OFF if the coil receives power flow. The reference remains OFF until set ON by a SET coil of the same reference (e.g., E1 in the example below). The last-solved SET coil or RESET coil of a pair takes precedence.

Note

RESET coils write an undefined result to the transition bit for the given reference. ***Do not use transition contacts on references used on RESET coils.***

Example:

In the following example, the coil represented by E1 is turned ON whenever reference E2 or E6 is ON. The coil represented by E1 is turned OFF whenever reference E5 or E3 is ON.

```
|
|   E2                                                              E1
|──| |──────────────────────────────────────────────────────────(S)─
|          |
|   E6     |
|──| |──|
|
|   E5                                                              E1
|──| |──────────────────────────────────────────────────────────(R)─
|          |
|   E3     |
|──| |──|
|
```
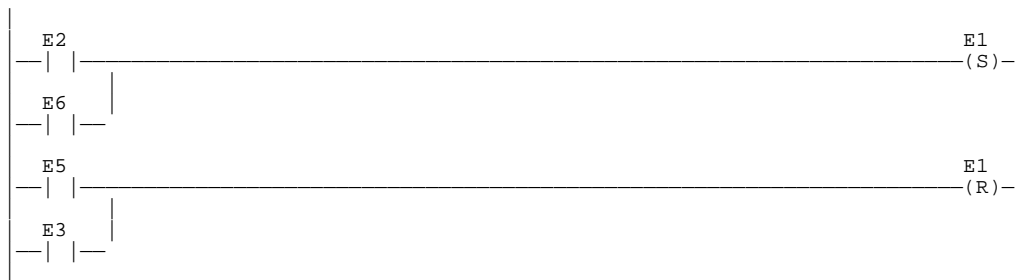
Retentive SET Coil   –(SM)–

Retentive SET and RESET coils are similar to SET and RESET coils, but they are retained across power failure or when the PLC transitions from **STOP** to **RUN** mode. A retentive SET coil sets a discrete reference ON if the coil receives power flow. The reference remains ON until reset by a retentive RESET coil.

Note

Retentive SET coils write an undefined result to the transition bit for the given reference. Do not use transition contacts on references used on retentive SET coils. (Refer to the information on "Transitions and Overrides" in chapter 2.)

Retentive RESET Coil   –(RM)–

This coil sets a discrete reference OFF if it receives power flow. The reference remains OFF until set by a retentive SET coil. The state of this coil is retained across power failure or when the PLC transitions from **STOP** to **RUN** mode.

Note

Retentive RESET coils write an undefined result to the transition bit for the given reference. Do not use transition contacts on references used on retentive RESET coils. (Refer to the information on "Transitions and Overrides" in chapter 2.)

Links

Horizontal and vertical links are used to connect elements of a line of ladder logic between functions. Their purpose is to complete the flow of logic ("power") from left to right in a line of logic.

Example:

In the following example, two horizontal links are used to connect contacts E2 and E5. A vertical link is used to connect contacts E3, E6, E7, E8, and E9 to E2.

```
|
|   E2                              E5                                          E1
—| |——————————————+—| |————————————————————————————————————(/)—
|                                   |
|   E3      E6      E7              |
—| |—+——|/|—+——| |—+
|        |   E8      |   E9      |
|        +——| |—+——| |—+
```
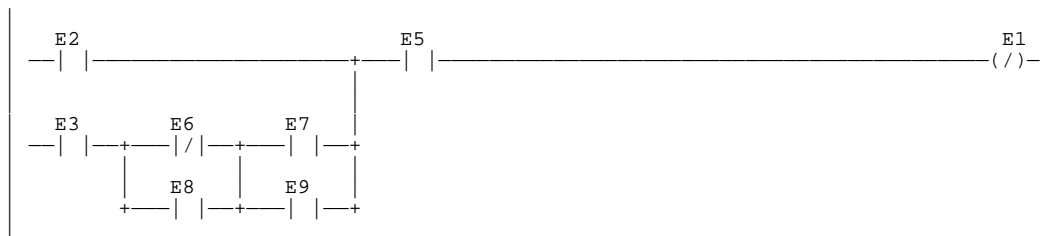
Continuation Coils (– – –<+>) and Contacts (<+>– – –)

Continuation coils (– – –<+>) and contacts (<+>– – –) are used to continue relay ladder logic beyond the limit of ten columns. The state of the last executed continuation coil is the flow state that will be used on the next executed continuation contact. If the flow of logic does not execute a continuation coil before it executes a continuation contact, the state of the contact will be no flow. The state of the continuation contact is cleared when the PLC transitions from **Stop** to **Run**, and there will be no flow unless the transition coil has been set since going to **Run** mode.

There can be only one continuation coil and contact per rung; the continuation contact must be in column 1 and the continuation coil must be in column 10.

Note

The Continuation coil and contact are not re-entrant and must not be used in different interrupt blocks.

Chapter 5

Timers and Counters

This chapter explains how to use on-delay, off-delay, and stopwatch-type timers, up counters, and down counters. The data associated with these functions is retentive through power cycles.

Abbreviation	Function	Page
ONDTR	Retentive On-Delay Timer	5-3
OFDT	Off-Delay Timer	5- 6
TMR	Simple On-Delay Timer	5- 9
UPCTR	Up Counter	5-12
DNCTR	Down Counter	5-14

Function Block Data Required for Timers and Counters

Each timer or counter uses three words (registers) of %R, %P, or %L memory to store the following information:

current value (CV)	word 1
preset value (PV)	word 2
control word	word 3

Note

Do not allow other functions to write to these registers when using the timer function.

When you enter a timer or counter, you must enter an address for the location of these three words (registers) directly below the graphic representing the function. For example:

```
                        _____
          (enable) ─ █ ONDTR █ ─ Q
                       │      │
                       │1.00s │
           (reset) ─── R      │
                       │      │
    (preset value) ─── PV CV ─── (current value)
                       ‾‾‾‾‾‾

                     (address)
```
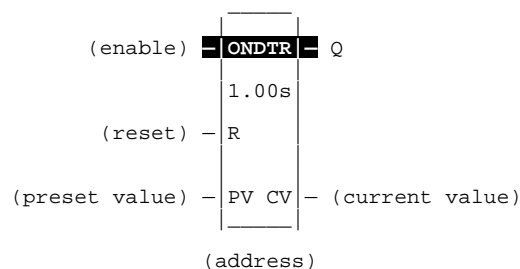
The control word stores the state of the Boolean input and output of its associated function block, as shown in the following format:
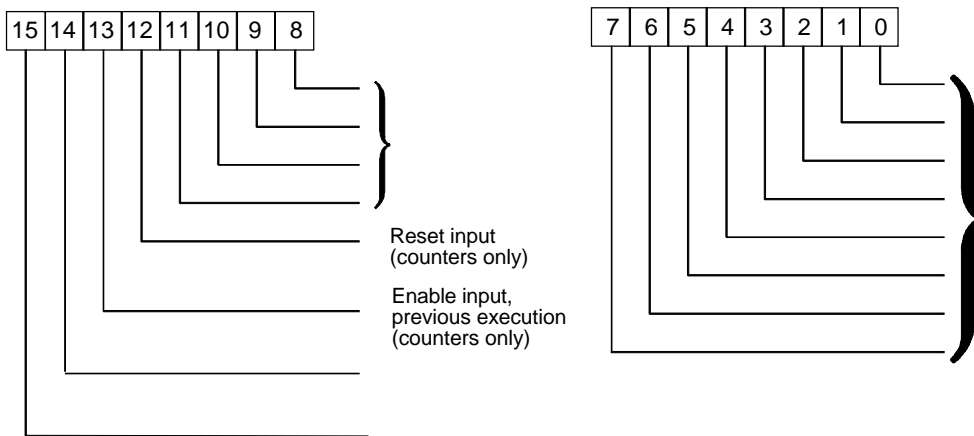
Bits 0 through 13 are used for timer accuracy; bits 0 through 10 are not used for counters.

ONDTR

A retentive on-delay timer (ONDTR) increments while it receives power flow and holds its value when power flow stops. Time may be counted in seconds (the default selection), tenths of seconds, or hundredths of seconds. 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 the ONDTR first receives power flow, it starts accumulating time (current value). When this timer is encountered in the ladder logic, its current value is updated.

Note

If multiple occurrences of the same timer with the same reference address are enabled during a CPU sweep, the current values of the times will be the same.

When the current value (CV) equals or exceeds the preset value (PV), output Q is energized. As long as the timer continues to receive power flow, it continues accumulating until the maximum value is reached. Once the maximum value is reached, it is retained and output Q remains energized regardless of the state of the enable input.
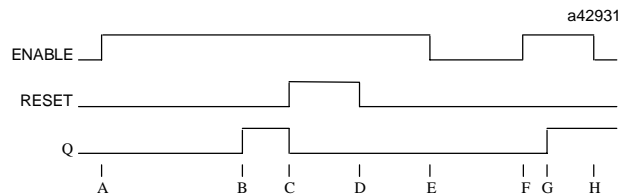

```
A  =    ENABLE goes high; timer starts accumulating.
B  =    CV reaches PV; Q goes high.
C  =    RESET goes high; Q goes low, accumulated time is reset.
D  =    RESET goes low; timer then starts accumulating again.
E  =    ENABLE goes low; timer stops accumulating.
        Accumulated time stays 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 current value becomes equal to the maximum time.
H  =    ENABLE goes low; timer stops accumulating time.
```

When power flow to the timer stops, the current value stops incrementing and is retained. Output Q, if energized, will remain energized. When the function receives power flow again, the current value again increments, beginning at the retained value. When reset (R) receives power flow, the current value is set back to zero and output (Q) is de-energized unless PV equals zero.

If PV equals zero and the timer is enabled, the output of the timer will activate. Subsequent removal of enable or activation of reset will have no effect on the timer output; it will remain enabled.

```
                          _____
        (enable) ─■ ONDTR ■─ Q
                         |      |
                         | 1.00s|
                         |      |
        (reset) ─|R      |
                         |      |
 (preset value) ─|PV  CV|─ (current value)
                         |_____|

                        (address)
```
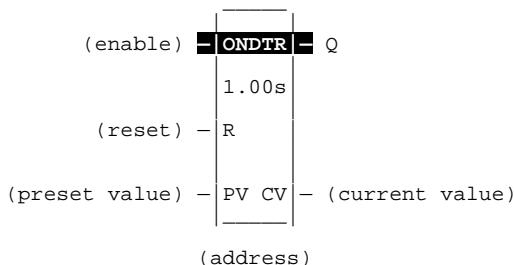
When the ONDTR is used in a program block that is not called every sweep, the timer accumulates time between calls to the program block unless it is reset. his means that it functions like a timer operating in a program with a much slower sweep than the timer in the main program block. For program blocks that are inactive for a long time, the timer 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. This time is applied to the timer when enabled, unless the timer is first reset.

Parameters:

Parameter	Description
address	The ONDTR uses three consecutive words (registers) of %R, %P, or %L memory to store the: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an ONDTR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. For more information, refer to page 4-13. **Note**: Do not use this address with other instructions. **Caution:** Overlapping references will result in erratic operation of the timer.
enable	When enable receives power flow, the timer's current value is incremented.
R	When R receives power flow, it resets the current value to zero.
PV	PV is the value to copy into the timer's preset value when the timer is enabled or reset. For a register (%R) PV reference, the PV parameter is specified as the second word of the address parameter. For example, an address parameter of %R00001 would use %R00002 as the PV parameter.
Q	Output Q is energized when the current value is greater than or equal to the preset value. Since no automatic initialization to the Q state occurs at power-up, the Q state is retentive across power failure.
CV	CV contains the current value of the timer.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
address									•	•	•			•		
enable	•															
R	•															
PV	•	•	•	•	•		•		•	•	•	•	•	•	•	•
Q	•															•
CV	•	•	•	•	•		•		•	•	•	•	•			•

•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, a retentive on-delay timer is used to create a signal (%Q00011) that turns on 8.0 seconds after %Q00010 turns on, and turns off when %Q00010 turns off.

```
|        _____
|%Q00010 |      |                                                        %Q00011
|—| |——— | ONDTR |————————————————————————————————————————————————————————( )—
|        |  0.1s |
|%Q00010 |       |
|—| / |——— | R    |
|        |       |
|        |       |
| CONST —| PV CV |—
| +00080 |_____|
|
|            %R00004
```
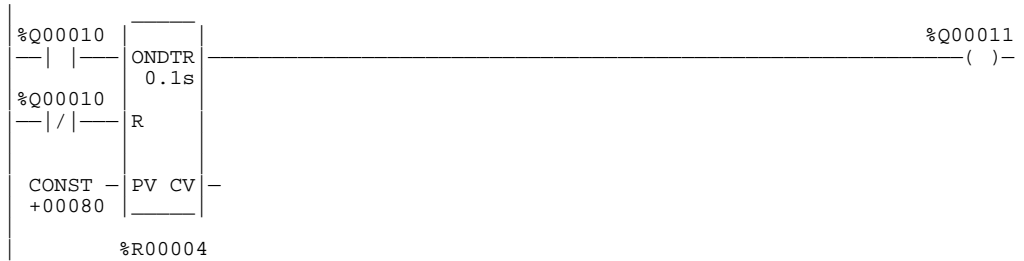
OFDT

The off-delay timer (OFDT) increments while power flow is off, and resets to zero when power flow is on. Time may be counted in seconds (the default selection), tenths of seconds, or hundredths of seconds. The range is 0 to +32767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

When the OFDT first receives power flow, it passes power to the right and the current value (CV) is set to zero. The output remains on as long as the function receives power flow. If the function stops receiving power flow from the left, it continues to pass power to the right and the timer starts accumulating time in CV.

Note

> If multiple occurrences of the same timer with the same reference address are enabled during a CPU sweep, the current values of the times will be the same.

The OFDT does not pass power flow if the preset value is zero or negative.

Each time the function is invoked with the enabling logic set OFF, the current value is updated to reflect the elapsed time since the timer was turned off. When the current value (CV) is equal to the preset value (PV), the function stops passing power flow to the right. The current value never exceeds the preset value.

When the function receives power flow again, the current value resets to zero.
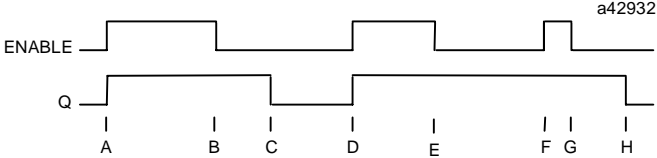

```
A  = ENABLE and Q both go high.
B  = ENABLE goes low; timer starts accumulating time.
C  = CV reaches PV; Q goes low, and timer stops accumulating time.
D  = ENABLE goes high; timer is reset (CV = 0).
E  = ENABLE goes low; timer starts accumulating time.
F  = ENABLE goes high; timer is reset (CV = 0).
G  = ENABLE goes low; timer begins accumulating time.
H  = CV reaches PV; Q goes low, and timer stops accumulating time.
```

```
                      _____
                     |      |
        (enable) ─ ■ | OFDT | ■ ─ Q
                     |      |
                     | 1.00s|
                     |      |
(preset value) ─ | PV CV | ─ (current value)
                     |      |
                     |_____|

                    (address)
```
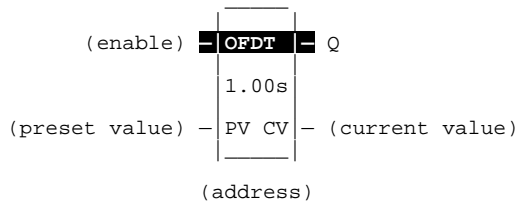
When the OFDT is used in a program block that is not called every sweep, the timer accumulates time between calls to the program block unless it is reset. This means that it functions like a timer operating in a program with a much slower sweep than the timer in the main program block. For program blocks that are inactive for a long time, the timer 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. This time is applied to the timer when enabled, unless the timer is first reset.

Parameters:

Parameter	Description
address	The OFDT uses three consecutive words (registers) of %R, %P, or %L memory to store the: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an OFDT, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. For more information, refer to page 4-13. **Note:** Do not use this address with other instructions. **Caution:** Overlapping references will result in erratic operation of the timer.
enable	When enable receives power flow, the timer's current value is incremented.
PV	PV is the value to copy into the timer's preset value when the timer is enabled or reset. For a register (%R) PV reference, the PV parameter is specified as the second word of the address parameter. For example, an address parameter of %R00001 would use %R00002 as the PV parameter.
Q	Output Q is energized when the current value is less than the preset value. Since no automatic initialization to the Q state occurs at power-up, the Q state is retentive across power failure.
CV	CV contains the current value of the timer.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
address									•	•	•			•		
enable	•															
PV	•	•	•	•	•		•		•	•	•	•	•	•	•	•
Q	•															•
CV	•	•	•	•	•		•		•	•	•	•	•	•		•

•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, an OFDT timer is used to turn off an output (%Q00001) whenever an input (%I00001) turns on. The output is turned on again 0.3 seconds after the input goes off.

```
        ____
%I00001 |————|                                              %Q00001
—| |———| OFDT |————————————————————————————————————————————( / )—
        | 0.10s|
        |      |
 CONST —| PV CV|—
 +00003 |_____|

          %P00019
```

TMR

The simple on-delay timer (TMR) function increments while it receives power flow and resets to zero when power flow stops. Time may be counted in seconds (the default selection), tenths of seconds, or hundredths of seconds. 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 the TMR receives power flow, the timer starts accumulating time (current value). The current value is updated when it is encountered in the logic to reflect the total elapsed time since the timer was last reset.

Note

If multiple occurrences of the same timer with the same reference address are enabled during a CPU sweep, the current values of the times will be the same. Additionally, a TMR timer will expire (pass power flow to the right) the first sweep that it is enabled if the previous sweep time was greater than the preset value (PV).

This update occurs as long as the enabling logic remains ON. When the current value (CV) equals or exceeds the preset value (PV), the function begins passing power flow to the right. The timer continues accumulating time until the maximum value is reached. When the enabling parameter transitions from ON to OFF, the timer stops accumulating time and the current value is reset to zero.
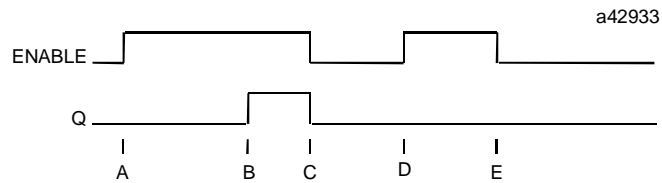

```
A  = ENABLE goes high; timer begins accumulating time.
B  = CV reaches PV; Q goes high, and timer continues accumulating time.
C  = ENABLE goes low; Q goes low; timer stops accumulating
     time and CV is cleared.
D  = ENABLE goes high; timer starts accumulating time.
E  = ENABLE goes low before CV reaches PV; Q remains low;
     timer stops accumulating time and is cleared to zero.
```

```
                      (enable) ─  TMR  ─ Q

                                │1.00s│
                (preset value) ─│PV CV│─ (current value)

                              (address)
```
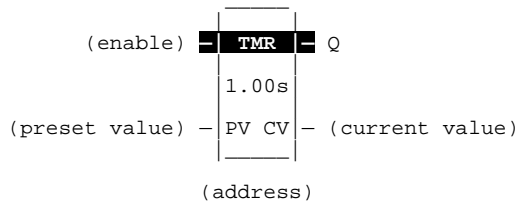
When the TMR is used in a program block that is not called every sweep, the timer accumulates time between calls to the program block unless it is reset. This means that it functions like a timer operating in a program with a much slower sweep than the timer in the main program block. For program blocks that are inactive for a long time, the timer 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. This time is applied to the timer when enabled, unless the timer is first reset.

Parameters:

Parameter	Description
address	The TMR uses three consecutive words (registers) of %R, %P, or %L memory to store the: • Current value (CV)　　= word 1. • Preset value (PV)　　= word 2. • Control word = word 3. When you enter an TMR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. For more information, refer to page 4-13. **Note:** Do not use this address with other instructions. **Caution:** Overlapping references will result in erratic operation of the timer.
enable	When enable receives power flow, the timer's current value is incremented. When the TMR is not enabled, the current value is reset to zero and Q is turned off.
PV	PV is the value to copy into the timer's preset value when the timer is enabled or reset. For a register (%R) PV reference, the PV parameter is specified as the second word of the address parameter. For example, an address parameter of %R00001 would use %R00002 as the PV parameter.
Q	Output Q is energized when TMR is enabled and the current value is greater than or equal to the preset value.
CV	CV contains the current value of the timer.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
address									•	•	•			•		
enable	•															
PV	•	•	•	•	•		•		•	•	•	•	•	•	•	•
Q	•															•
CV	•	•	•	•	•		•		•	•	•	•	•	•		•

•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, an on-delay timer TMRID is used to control the length of time that coil DWELL is on. When the 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 one-half 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.

```
|
|  DO_DWL   REL                                                              DWELL
|—| |—+—| / |————————————————————————————————————————————————————————————( )—
|      |
|  DWELL |
|—| |—+
|                _____
|  DWELL       |     |                                                        REL
|—| |—————| TMR |————————————————————————————————————————————————————————( )—
|              | 0.1s|
|              |     |
|  CONST —| PV CV |—
|  +00005      |_____|
|
|                TRMID
```
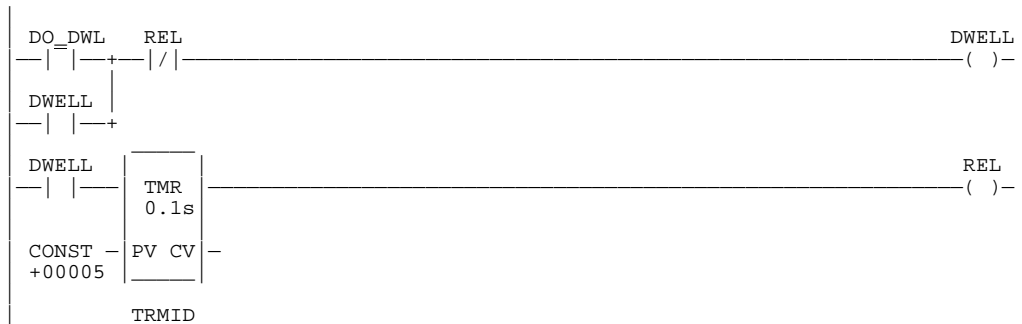
UPCTR

The Up Counter (UPCTR) function is used to count up to a designated value. The range is 0 to +32,767 counts. When the up counter reset is ON, the current value (CV) of the counter is reset to 0. Each time the enable input transitions from OFF to ON, the current value is incremented by 1. The current value (CV) can be incremented past the preset value (PV). The output is ON whenever the current value is greater than or equal to the preset value.

The output state (Q) of the UPCTR is retentive on power failure; no automatic initialization occurs at power-up.

```
    (enable) ─ UPCTR ─   (Q)

     (reset) ─ R

(preset value) ─ PV CV ─ (current value)

               (address)
```

Parameters:

Parameter	Description
address	The UPCTR uses three consecutive words (registers) of %R, %P, or %L memory to store the: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an UPCTR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. For more information, refer to page 4-13. **Note:** Do not use this address with another up counter, down counter, or any other instruction or improper operation will result. **Caution:** Overlapping references will result in erratic operation of the counter.
enable	On a positive transition of enable, the current count is incremented by one.
R	When R receives power flow, it resets the current value back to zero.
PV	PV is the value to copy into the counter's preset value when the counter is enabled or reset. For a register (%R) PV reference, the PV parameter is specified as the second word of the address parameter. For example, an address parameter of %R00001 would use %R00002 as the PV parameter.
Q	Output Q is energized when the current value is greater than or equal to the preset value. The Q state is retentive on power failure; no automatic initialization occurs at power-up.
CV	CV contains the current value of the counter.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
address									•	•	•			•		
enable	•															
R	•															
PV	•	•	•	•	•		•		•	•	•	•	•	•	•	•
Q	•															•
CV	•	•	•	•	•		•		•	•	•	•	•	•		•

•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, every time input %I00012 transitions from OFF to ON, up counter PRT_CNT counts up by 1; internal coil %M00001 is energized whenever 100 parts have been counted. Whenever %M00001 is ON, the accumulated count is reset to zero.

```
                 _____
%I00012   |      |      |                                              %M00001
——| |—— >UPCTR —————————————————————————————————————————————————————————( )—
          |      |      |
%M00001   |      |      |
——| |—————|      |R     |
          |      |      |
          |      |      |
  CONST — |PV  CV|—
  +00100  |_____|
                PRT_CNT
```
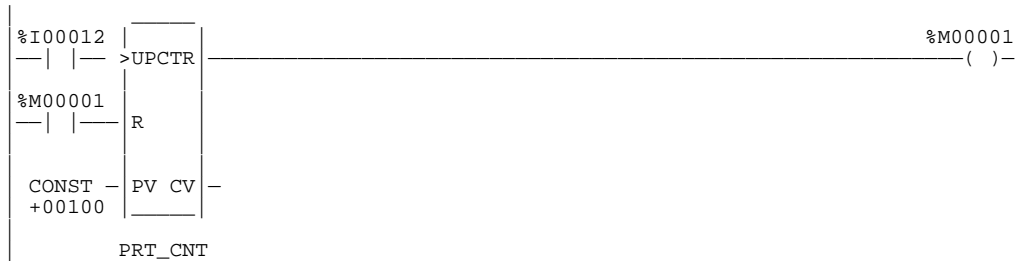
DNCTR

The Down Counter (DNCTR) function is used to count down to zero from a preset value. The minimum preset value is zero; the maximum preset value is +32,767 counts. The minimum current value is –32,768. When reset, the current value (CV) of the counter is set to the preset value (PV). When the enable input transitions from OFF to ON, the current value decrements by one. The output is ON whenever the current value is less than or equal to zero.

The output state (Q) of the DNCTR is retentive on power failure; no automatic initialization occurs at power-up.

```
                          _____
                         |     |
        (enable)  ->DNCTR |  -  (Q)
                         |     |
        (reset)   —  R   |
                         |     |
  (preset value)  —  PV CV —  (current value)
                         |_____|

                       (address)
```

Parameters:

Parameter	Description
address	The DNCTR uses three consecutive words (registers) of %R, %P, or %L memory to store the: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an DNCTR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. For more information, refer to page 4-13. **Note:** Do not use this address with another down counter, up counter, or any other instruction or improper operation will result. **Caution:** Overlapping references will result in erratic operation of the counter.
enable	On a positive transition of enable, the current value decrements by one.
R	When R receives power flow, it resets the current value to the preset value.
PV	PV is the value to copy into the counter's preset value when the counter is enabled or reset. For a register (%R) PV reference, the PV parameter is specified as the second word of the address parameter. For example, an address parameter of %R00001 would use %R00002 as the PV parameter.
Q	Output Q is energized when the current value is less than or equal to zero. The Q state is retentive on power failure; no automatic initialization occurs at power-up.
CV	CV contains the current value of the counter.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
address									•	•	•			•		
enable	•															
R	•															
PV	•	•	•	•	•		•		•	•	•	•	•	•	•	•
Q	•															•
CV	•	•	•	•	•		•		•	•	•	•	•			•

•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the down counter identified as COUNTP counts 5000 new parts before energizing output %Q00005.

```
 |        _____
 |NEW_PRT |      |                                                    %Q00005
 |—| |— >DNCTR |————————————————————————————————————————————————————————( )—
 |        |      |
 |NXT_BAT |      |
 |—| |——— R      |
 |        |      |
 |        |      |
 | CONST —|PV CV |—
 | +05000 |_____|
 |
 |           COUNTP
```
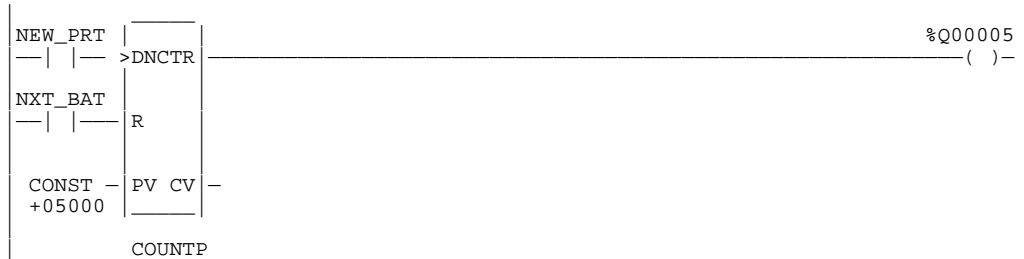
Up/Down Counter Pair Example:

In the following example, the PLC is used to keep track of the number of parts contained in a temporary storage area. There are two ways of accomplishing this function using the Series 90-70 instruction set.

The first method is to use 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. This is the preferred method for this application.

```
|
|   << RUNG 4 >>
|
|
|%I00003
|—| |—
|            _____
|%I00001| |
|—| |— >UPCTR|——————————————————————————————————  %Q00001
|       |1.00s|                                          —( )—
|%I00009|     |
|—| |——|R    |
|       |     |
|       |     |
| CONST—|PV CV|—%R00104
| +00005|_____|
|
|          %R00100
|
|
|   << RUNG 5 >>
|
|
|%I00003
|—| |—
|            _____
|%I00002| |
|—| |— >DNCTR|——————————————————————————————————  %Q00002
|       |     |                                          —( )—
|ALW_OFF|     |
|—| |——|R    |
|       |     |
|       |     |
| CONST—|PV CV|—%R00100
| +00005|     |
|       |_____|
|
|          %R00104
|
```

The second method, shown below, uses the ADD and SUB functions to provide storage tracking.

```
|
| << RUNG 7 >>
|
|              _____
|%I00004     |      |
|————|↑|——————| ADD_ |——
|             | INT  |
|             |      |
|%R00201——   |I1   Q|——%R00201
|             |      |
|             |      |
|  CONST —   |I2    |
|  +00001    |_____|
|
|
| << RUNG 8 >>
|
|       _____
|%I00005 |   |
|————|↑|——————| SUB_|——
|      | INT |
|      |   |
|%R00201——|I1  Q|——%R00201
|      |   |
|      |   |
| CONST —|I2  |
| +00001 |_____|
|
```
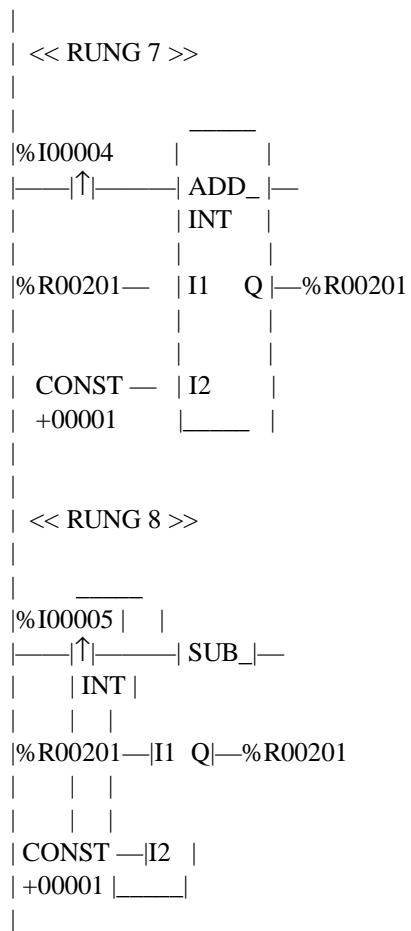
Math Functions

This chapter describes the Logicmaster 90 math functions:

Abbreviation	Function	Description	Page
ADD	Addition	Add two numbers.	6-2
SUB	Subtraction	Subtract one number from another.	6-2
MUL	Multiplication	Multiply two numbers.	6-2
DIV	Division	Divide one number by another, yielding a quotient.	6-2
MOD	Modulo Division	Divide one number by another, yielding a remainder.	6-4
SQRT	Square Root	Find the square root of an integer or real value.	6-6
ABS	Absolute Value	Find the absolute value of an integer, double precision integer, or real value.	6-8
SIN, COS, TAN, ASIN, ACOS, ATAN	Trigonometric Functions	Perform the appropriate function on the real value in input IN.	6-10
LOG, LN EXP, EXPT	Logarithmic/Exponential Functions	Perform the appropriate function on the real value in input IN.	6-12
RAD, DEG	Radian Conversion	Perform the appropriate function on the real value in input IN.	6-14

Note

Division and modulo division are similar functions which differ in their output; division finds a quotient, while modulo division finds a remainder.

MATH   (ADD, SUB, MUL, DIV)

Math functions include addition, subtraction, multiplication, and division. When a function receives power flow, the appropriate math function is performed on input parameters I1 and I2. These parameters must be the same data type. Output Q is the same data type as I1 and I2.

Note

DIV rounds down; it does not round to the closest integer.
(For example, 24 DIV 5 = 4.)

Math functions operate on these types of data:

Data Type	Description
INT	Signed integer.
UINT	Unsigned integer.
DINT	Double precision signed integer.
REAL	Floating point.
MIXED	Mixed is available for MUL and DIV only.

Note

MUL_MIXED inputs are the same as INT; MUL_MIXED output is the same as DINT. DIV_MIXED input I1 is the same as DINT; DIV_MIXED input I2 and outputs are the same as INT.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, refer to chapter 2, section 3, "Program Organization and User Data."

If the operation of INT or DINT operands 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. If the operation of UINT operands results in overflow, the output reference is set to the overflow value.

If the operation does not result in overflow, the ok output is set ON unless one of these invalid REAL operations occurs:

- For ADD, $(+\infty)+(-\infty)$ For SUB, $(\pm\infty)-(\pm\infty)$

- For MUL, $0\times(\infty)$

- For DIV, 0 divided by 0.

- For DIV, ∞ divided by ∞

- I1 and/or I2 is NaN (Not a Number).

In these cases, ok is set OFF.
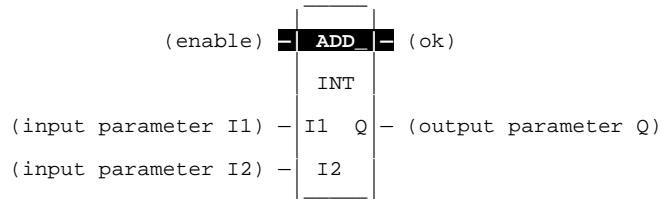
```
                          _____
               (enable) ─▌ ADD_ ▐─ (ok)
                         │      │
                         │ INT  │
(input parameter I1) ─── I1   Q ── (output parameter Q)
                         │      │
(input parameter I2) ─── I2     │
                         │_____│
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first value used in the operation. (I1 is on the left side of the mathematical equation, as in I1 – I2.)
I2	I2 contains a constant or reference for the second value used in the operation. (I2 is on the right side of the mathematical equation, as in I1 – I2.)
ok	The ok output is energized when the function is performed without dividing by zero, unless an invalid operation occurs or I1 and/or I2 is NaN.
Q	Output Q contains the result of the operation.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o		o		•	•	•	•	•	•	•	
I2	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o		o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for UINT or INT data only; not valid for DINT or REAL.

Note

For restrictions within a parameterized subroutine block, refer to the "Restrictions on Formal Parameters within a Parameterized Subroutine Block" section of Chapter 2.

Example:

In the following example, whenever input %I00001 is set, the integer content of %R00002 is decremented by 1 and coil %Q00001 is turned on, provided there is no overflow in the subtraction.

```
                   _____
│%I00001          │      │                                                      %Q00001
─┤ ├──────        │ SUB_ │──────────────────────────────────────────────────────( )─
│                 │ INT  │
│                 │      │
│%R00002───       I1   Q │─%R00002
│ +00095          │      │
│                 │      │
│ CONST ───       I2     │
│ +00001          │_____│
│
```
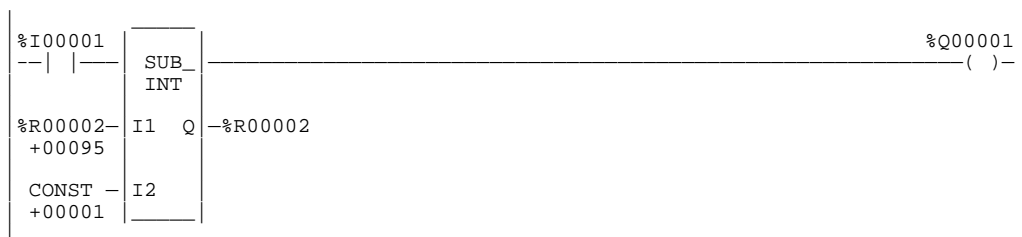
MOD   (INT, UINT, DINT)

The Modulo (MOD) function is used to divide one value by another value of the same data type, to obtain a remainder. If signed or double precision signed integers are used, the sign of the result is always the same as the sign of I1.

The MOD function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
UINT	Unsigned integer.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, refer to chapter 2, section 3, "Program Organization and User Data."

When the function receives power flow, it divides input parameter I1 by input parameter I2. These parameters must be the same data type. Output Q is calculated using the formula:

```
Q = I1 - ((I1 DIV I2) * I2)
```

where DIV produces an integer number. Q is the same data type as input parameters I1 and I2.

OK is always ON when the function receives power flow, unless there is an attempt to divide by zero. In that case, it is set OFF.
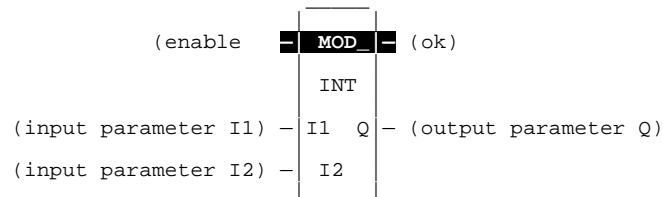
```
                          _____
        (enable     ─  | MOD_ | ─  (ok)
                       |_____|
                       | INT  |
(input parameter I1) ─ | I1  Q| ─  (output parameter Q)
(input parameter I2) ─ | I2   |
                       |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the value to be divided by I2.
I2	I2 contains a constant or reference for the value to be divided into I1.
ok	The ok output is energized when the function is performed without dividing by zero.
Q	Output Q contains the result of dividing I1 by I2 to obtain a remainder.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o		o		•	•	•	•	•	•	•	
I2	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o		o		•	•	•	•	•	•		

<u>Note</u>: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for UINT or INT data only; not valid for DINT.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the remainder of the integer division of BOXES into PALLETS is placed into NT_FULL whenever %I00001 is ON.

```
|         _____
|%I00001 |      |
|—| |——— | MOD_ | —
|        | INT  |
|        |      |
|PALLETS—| I1  Q| —NT_FULL
|—00017  |      |   t00005
|        |      |
| BOXES —| I2   |
| +00006 |_____|
|
```

SQRT   (INT, DINT, REAL)

The Square Root (SQRT) function is used to find the square root of a value. When the function receives power flow, the value of output Q is set to the integer portion of the square root of the input IN. The output Q must be the same data type as IN.

The SQRT function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer
REAL	Floating point.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, refer to chapter 2, section 3, "Program Organization and User Data."

OK is set ON if the function is performed without overflow, unless one of these invalid REAL operations occurs:

- $IN < 0$.

- IN is NaN (Not a Number).

Otherwise, ok is set OFF.
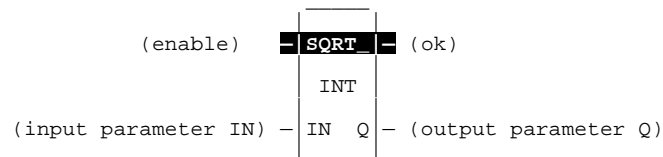
```
                           _____
        (enable)       ─ SQRT_ ─  (ok)
                          │       │
                          │  INT  │
                          │       │
(input parameter IN) ─│ IN  Q │─ (output parameter Q)
                          │_____│
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains a constant or reference for the value whose square root is to be calculated. If IN is less than zero, the function will not pass power flow.
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs or IN is NaN.
Q	Output Q contains the square root of IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o		o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.
o Valid reference for INT only; not valid for DINT or REAL.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the square root of the integer number located at %AI0001 is placed into the result located at %R00003 whenever %I00001 is ON.

```
 |             _____
 |%I00001     |     |
 |—| |——|     | SQRT_|
 |            | INT |
 |            |     |
 |%AI0001—| IN  Q |—%R00003
 |            |_____|
 |
```

ABS (INT, DINT, REAL)

The Absolute Value (ABS) function is used to find the absolute value of an integer, double precision integer, or real value. When the function receives power flow, it places the absolute value of input IN in output Q.

The ABS function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer
REAL	Floating point.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, refer to chapter 2, section 3, "Program Organization and User Data."

The ok output will receive power flow, unless one of the following conditions occurs:

- For INT type, IN is MININT.

- For DINT type, IN is MINDINT.

- For REAL type, IN is NaN (Not a Number).
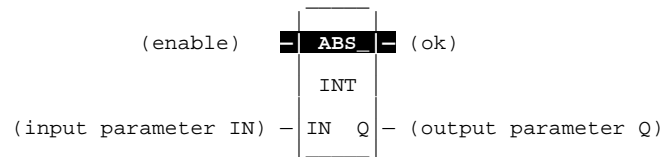
```
                              ┌─────┐
        (enable)        ─┤ ABS_ ├─ (ok)
                              │      │
                              │ INT  │
                              │      │
(input parameter IN) ─┤ IN  Q├─ (output parameter Q)
                              └─────┘
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the integer or real value to be operated on.
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs and/or IN is NaN.
Q	Output Q contains the absolute value of IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o		o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT only; not valid for DINT or REAL.

Note

For restrictions within a parameterized subroutine block, refer to the "Restrictions on Formal Parameters within a Parameterized Subroutine Block" section of Chapter 2..

Example:

In the following example, the absolute value of –2976, which is 2976, is placed in %R00010.

```
               _____
              |      |
|─────────────| ABS_ |─
              | INT  |
              |      |
  CONST ──────| IN  Q|──%R00010
  -2976       |_____|    2976
```

Trig Functions   (SIN, COS, TAN, ASIN, ACOS, ATAN)

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

The ASIN, ACOS, and ATAN functions are used to find the inverse sine, cosine, and tangent, respectively, of its input. When one of these functions receives power flow, it computes the inverse sine (or cosine or tangent) of IN and stores the result in output Q, whose units are radians. Both IN and Q are floating-point values.

The SIN, COS, and TAN functions accept a broad range of input values, where $-2^{63} < IN < +2^{63}$, ($2^{63} \approx 9.22 \times 10^{18}$).

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

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

The ACOS_REAL function will produce a result Q such that:

$$ \text{ACOS (IN)} = \quad - 0 \le Q \le \pi $$

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_REAL function will produce a result Q such that:

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

```
                        _____
        (enable)   ─┤ SIN_ ├─ (ok)
                     │ REAL │
(input parameter IN) ─┤ IN  Q ├─ (output parameter Q)
                     └──────┘
```
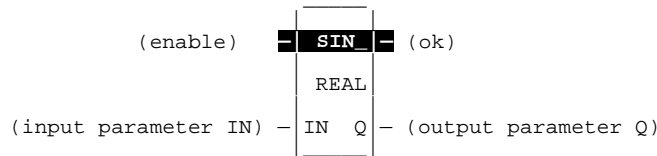
NOTE

These functions are only available on floating-point versions of the Series 90-70 CPU.

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the real value to be operated on.
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs and/or IN is NaN.
Q	Output Q contains the trigonometric value of IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•								•	•	•	•	•	•	•	
ok	•															•
Q	•								•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the COS of the value in %P00001 is placed in %P00033.

```
|                     |‾‾‾‾‾|
|――――――――――――――――| COS_|―
|                     |REAL |
|         %P00001―|IN  Q|―%P00033
|         +3.141500|‾‾‾‾‾| œ1.000000
|
```
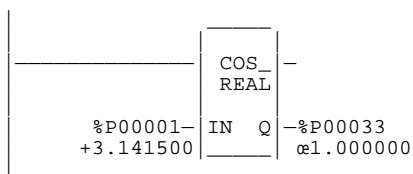
Logarithmic/Exponential Functions   (LOG, LN, EXP, EXPT)

The LOG, LN, and EXP functions have two input parameters and two output parameters. When the function receives power flow, it performs the appropriate logarithmic/exponential operation on the real value in input IN and places the result in output Q.

- For the LOG function, the base 10 logarithm of IN is placed in Q.

- For the LN function, the natural logarithm of IN is placed in Q.

- For the EXP function, *e* is raised to the power specified by IN and the result is placed in Q.

The EXPT function has three input parameters and two output parameters. When the function receives power flow, the value of input I1 is raised to the power specified by the value I2 and the result is placed in output Q.

The ok output will receive power flow, unless IN is NaN (Not a Number) or is negative.

```
                        ┌─────┐
        (enable)    ─┤ LOG_ ├─  (ok)
                        │ REAL │
(input parameter IN) ─┤ IN  Q ├─ (output parameter Q)
                        └─────┘
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the real value to be operated on.
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs and/or IN is NaN or is negative.
Q	Output Q contains the logarithmic/exponential value of IN.

Note

These functions are only available on floating-point versions of the Series 90-70 CPU.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN*	•								•	•	•	•	•	•	•	
ok	•															•
Q	•								•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
*   For the EXPT function, input IN is replaced by input parameters I1 and I2.
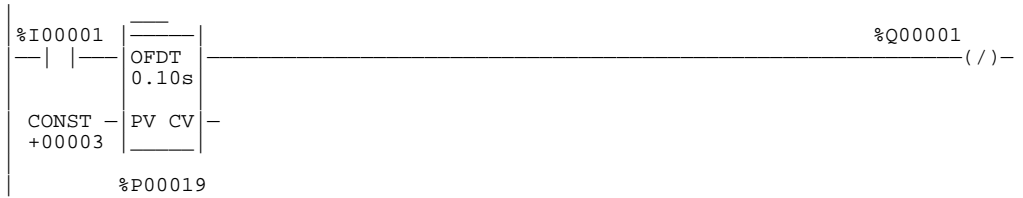•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a
parameterized subroutine block, refer to "Restrictions on Formal Parameters
Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the value of %AI0001 is raised to the power of 2.5 and the result is
placed in %R00001.

```
                          _____
                        | EXPT_ | ─
                        |  REAL |
                        |       |
            %AI0001─────| I1  Q |─%R00001
                        |       |
                        |       |
             CONST ─────| I2    |
           2.50000E+00  |_____|
```
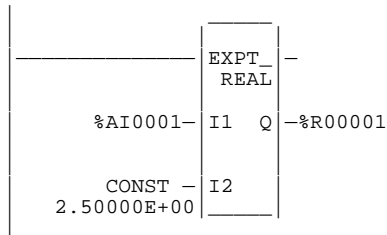
Radian Conversion   (RAD, DEG)

When the function receives power flow, the appropriate conversion (RAD_TO_DEG or DEG_TO_RAD) is performed on the real value in input IN and the result is placed in output Q.

The ok output will receive power flow unless IN is NaN (Not a Number).

```
                                     _____
                                    |      |
        (enable) ━┫  RAD_ ┣━ (ok)
                                    | TO_  |
                                    | DEG  |
(input parameter IN) ─│IN  Q│─ (output parameter Q)
                                    |_____|
```
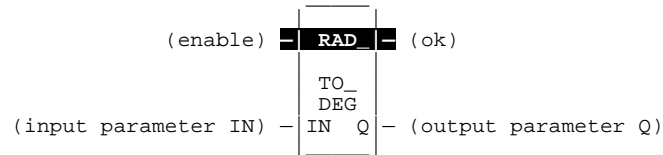
Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the real value to be operated on.
ok	The ok output is energized when the function is performed without overflow, unless IN is NaN.
Q	Output Q contains the converted value of IN.

Note

These functions are only available on floating-point versions of the Series 90-70 CPU.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•								•	•	•	•	•	•	•	
ok	•															•
Q	•								•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, +1500 is converted to DEG and is placed in %R0001.

```
                                          _____
                                         |     |
 _____| RAD_|
                                         | TO_ |
                                         | DEG |
                           CONST         |     |
                       +1500.000 ——————— | IN Q|——— %R0001
                                         |_____|   85993.67
```
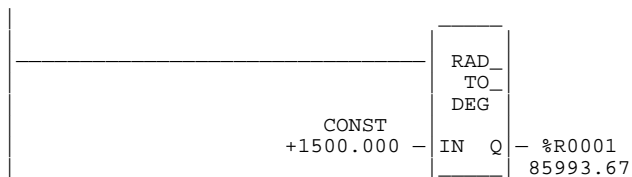
Chapter 7

Relational Functions

Relational functions are used to compare two numbers. This chapter describes the following relational functions:

Abbreviation	Function	Description	Page
EQ	Equal	Test two numbers for equality.	7-2
NE	Not Equal	Test two numbers for non-equality.	7-2
GT	Greater Than	Test for one number greater than another.	7-2
GE	Greater Than or Equal	Test for one number greater than or equal to another.	7-2
LT	Less Than	Test for one number less than another.	7-2
LE	Less Than or Equal	Test for one number less than or equal to another.	7-2
CMP	Compare	Test for one number less than, equal to, or greater than another.	7-4
RANGE	Range	Determine whether a number is within a specified range.	7-6

EQ, NE, GT, GE, LT, and LE (INT, UINT, DINT, REAL)

Relational functions are used to determine the relation of two values. When the function receives power flow, it compares input parameter I1 to input parameter I2. These parameters must be the same data type.

Relational functions operate on these types of data:

Data Type	Description
INT	Signed integer.
UINT	Unsigned integer.
DINT	Double precision signed integer.
REAL	Floating point.

The default data type is signed integer. To compare either signed integer, unsigned integer, or double precision integers, select the new data type after selecting the relational function. To compare data of other types or of two different types, first use the appropriate conversion function (described in Chapter 12, *Conversion Functions*) to change the data to one of the integer types.

If input parameters I1 and I2 match the specified relation, output Q receives power flow and is set ON (1); otherwise, it is set OFF (0).

Output ok will always receive power flow when the function is enabled, unless I1 and/or I2 is NaN (Not a Number).

```
                            ┌──────┐
           (enable)  ─┤ EQ_ ├─      (ok)
                        │      │
                        │ REAL │
(input parameter I1) ─┤ I1  Q ├─ (output parameter Q)
                        │      │
(input parameter I2) ─┤ I2    │
                        └──────┘
```
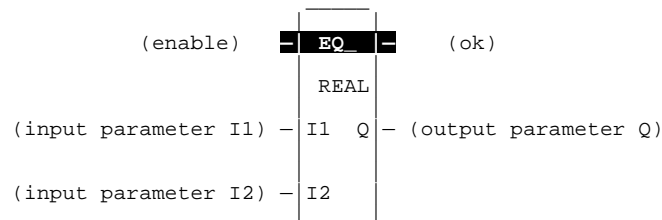
Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first value to be compared. (I1 is on the left side of the relational equation, as in I1 < I2).
I2	I2 contains a constant or reference for the second value to be compared. (I2 is on the right side of the relational equation, as in I1 < I2).
ok	The ok output is energized when the function is performed without error, unless I1 and/or I2 is NaN.
Q	Output Q is energized when I1 and I2 match the specified relation.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o		o		•	•	•	•	•	•	•	
I2	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o		o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT or UINT data only; not valid for DINT or REAL.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
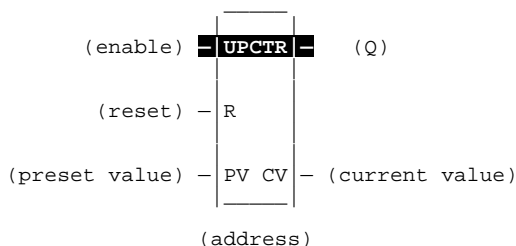
Example:

In the following example, two double precision signed integers, PRT_MDE and BIN_FUL, are compared whenever %I00001 is set. If PRT_MDE is less than BIN_FUL, coil %Q00002 is turned on.

```
  %I00001              _____
 ——| |————————————————|     |—
                       | LT_ |
                       | DINT|                                      %Q00002
            PRT_MDE—|I1   Q |——————————————————————————————————————( )—
                       |     |
            BIN_FUL—|I2     |
                       |_____|
```

CMP   (INT, UINT, DINT, REAL)

Use the Compare (CMP) function to test for one number less than, equal to, or greater than another.

When the function receives power flow, it compares the value I1 to the value I2. These values must be the same data type.

The CMP function operates on these types of data:

Data Type	Description
INT	Signed integer.
UINT	Unsigned integer.
DINT	Double precision signed integer.
REAL	Floating point.

The default data type is signed integer. To compare either signed integer, unsigned integer, or double precision signed integers, select the new data type after selecting the relational function. To compare data of other types or of two different types, first use the appropriate conversion function (described in Chapter 12, *Conversion Functions*) to change the data to one of the integer types.

LT, EQ, and GT are set ON (1) or OFF (0) as a result of the comparison.

Output ok will always receive power flow when the function is enabled, unless I1 and/or I2 is NaN (Not a Number).

```
                         _____
      (enable) ─█ CMP_ █─   (ok)

                        │ REAL │
(input parameter I1) ─│I1 LT │─ (output parameter for less than)

(input parameter I2) ─│I2 EQ │─ (output parameter for equal)

                      │   GT │─ (output parameter for greater than)
                         ──────
```
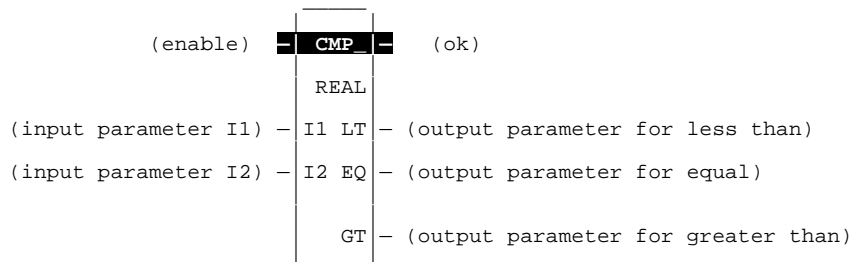
Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first value to be compared.
I2	I2 contains a constant or reference for the second value to be compared.
ok	The ok output is energized when the function is performed without error, unless I1 and/or I2 is NaN.
LT	Output LT is energized when I1 is less than I2.
EQ	Output EQ is energized when I1 is equal to I2.
GT	Output GT is energized when I1 is greater than I2.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o		o		•	•	•	•	•	•	•	
I2	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
LT	•															•
EQ	•															•
GT	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT or UINT data only; not valid for DINT or REAL.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
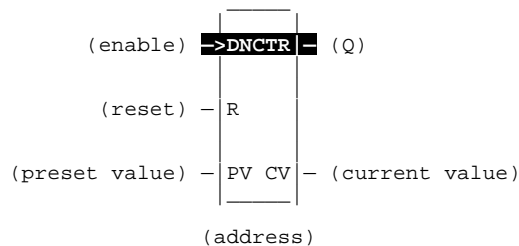
Example:

In the following 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.

```
 |    _____
 |%I00001  |
 |—| |——— CMP_ —
 |     | UINT |
 |     |      |                                            %M00001
 |SHIPS —|I1 LT————————————————————————————————————————————( )—
 |     |      |                                            %M00002
 |BOATS —|I2 EQ————————————————————————————————————————————( )—
 |     |      |                                            %M00003
 |     | GT————————————————————————————————————————————————( )—
 |     |_____|
 |
```
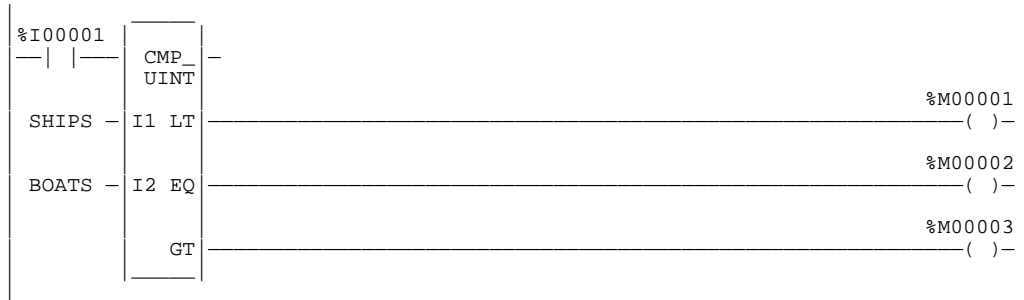
RANGE   (INT, UINT, DINT, WORD, DWORD)

The RANGE function is used to compare a single input value against two delimiters to determine whether the input value falls within the range of the delimiters.

Note

The Range function is only available on a Release 5 or higher CPU.

The RANGE function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
UINT	Unsigned integer.
WORD	Word data type.
DWORD	Double word data type.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, please refer to chapter 2, section 2, "Program Organization and User References/Data."

When the function is enabled, the RANGE function block will compare the value in input parameter IN against the range specified by the values of the two delimiters specified by parameters L1 and L2, inclusively. When the value in IN is within the range specified by L1 and L2, output parameter Q is set ON (1). Otherwise, Q is set OFF (0). If the operation is successful, the ok output will receive power flow.

```
                           _____
         (enable) ─ RANGE ─       (ok)
                          │ INT │
 (delimiter L1) ─│L1   Q│─ (output parameter Q)
 (delimiter L2) ─│L2    │
(value to be compared) ─│IN    │
                           ‾‾‾‾‾
```
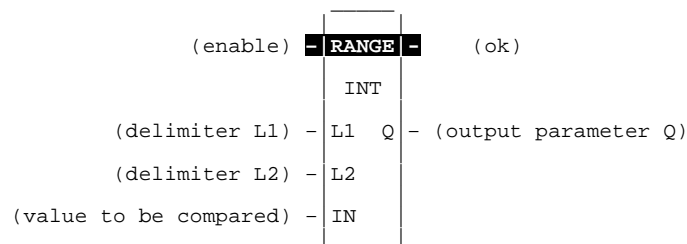
Note

Limit parameters L1 and L2 represent the endpoints of a range. There is no fixed minimum/maximum or high/low connotation assigned to either parameter. Thus, a desired range of 0 to 100 could be specified by assigning 0 to L1 and 100 to L2 or 0 to L2 and 100 to L1. In either case, a value of 45 for the input parameter IN would result in the output Q being set ON (1).

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
L1	L1 contains the start point of the range.
L2	L2 contains the end point of the range.
IN	IN contains the value to be compared against the range specified by L1 and L2.
ok	The ok output is energized unless an error is encountered.
Q	Output Q is energized when the value in IN is within the range specified by L1 and L2, inclusive.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
L1		o	o	o	o		o		•	•	•	•	•		•†	
L2		o	o	o	o		o		•	•	•	•	•		•†	
IN		o	o	o	o		o		•	•	•	•	•			
ok		o	o	o	o		o		•	•	•	•	•			
Q	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT ,WORD or UINT data only; not valid for DINT or DWORD.
†   Constants are limited to integer values for double precision signed integer operations.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
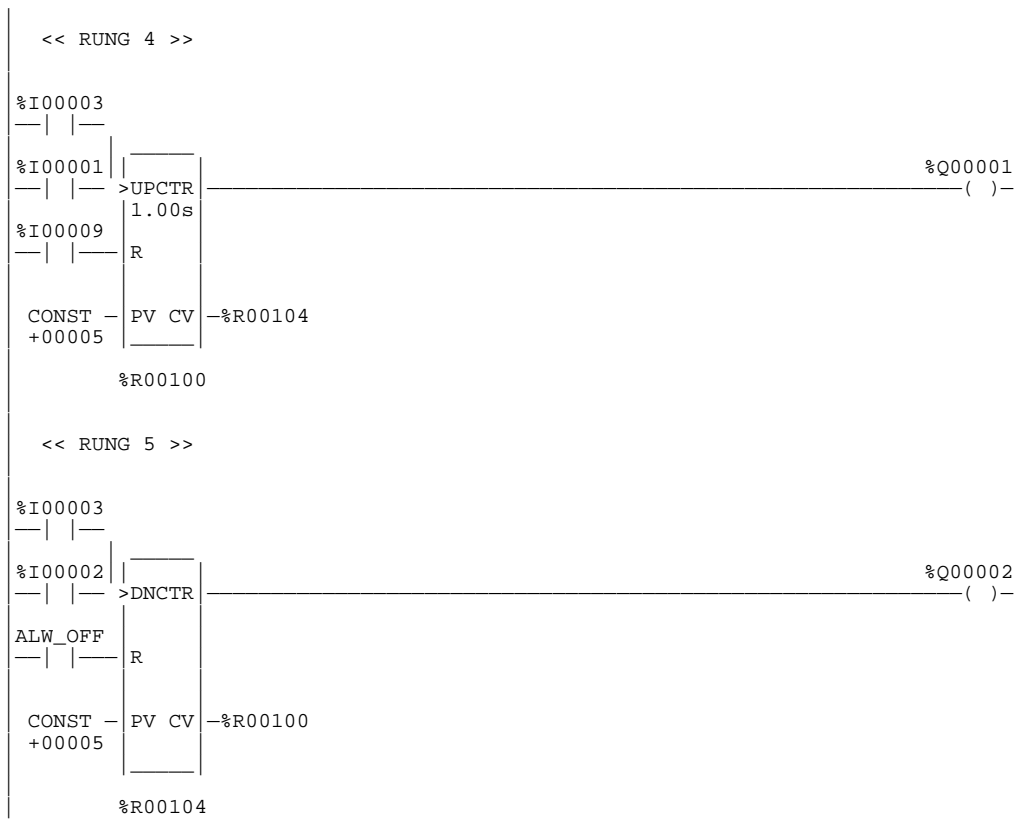
Example:

In the following example, the value in %R00001 is 10, and the value in %R00002 is 50. The output Q will be ON for all IN values in %R00003 greater than or equal to 10 and less than or equal to 50. Output Q will be OFF for all IN values in %R00003 less than 10 or greater than 50. The ok output is set ON.

```
       |  _____
%I00001 | |     |                                                      %M00001
 —| |—0+RANGE ————————————————————————————————————————————————————————( )—
       |  INT  |
       |       |                                                      %M00002
%R00001—|L1   Q|————————————————————————————————————————————————————————( )—
       |       |
%R00002—|L2     |
       |       |
%R00003—|IN     |
       |  _____|
       |
```
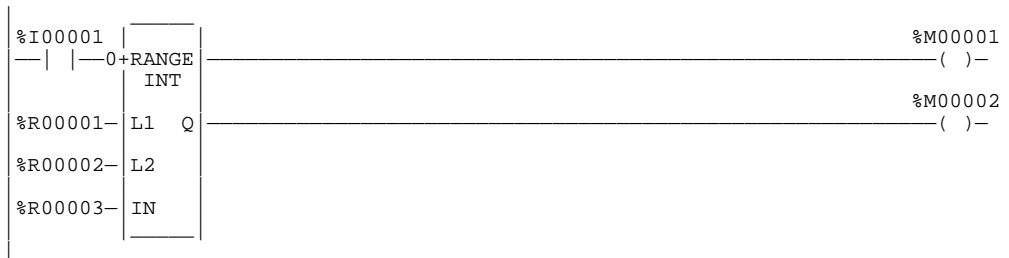
Bit Operation Functions

Bit operation functions perform comparison, logical, and move operations on bit strings. The maximum string length is 256 words or double words. Bit operation functions require WORD or DWORD data; the default data type is WORD.

Although data must be specified in 16-bit or 32-bit increments, these functions operate on data as a continuous string of bits, with bit 1 of the first word being the Least Significant Bit (LSB). The last bit of the last word is the Most Significant Bit (MSB). For example, if you specified three words of data beginning at reference %L00100, it would be operated on as 48 contiguous bits.

%L00100	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	← bit 1 (LSB)
%L00101	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	
%L00102	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	

↑

(MSB)

Note

Overlapping input and output reference address ranges in multi-word functions may produce unexpected results.

The following bit operation functions are described in this chapter:

Abbreviation	Function	Description	Page
AND	Logical AND	If a bit in bit string I1 and the corresponding bit in bit string I2 are both 1, place a 1 in the corresponding location in output string Q.	8-3
OR	Logical OR	If a bit in bit string I1 qnd/or the corresponding bit in bit string I2 are both 1, place a 1 in the corresponding location in output string Q.	8-3
XOR	Logical exclusive OR	If a bit in bit string I1 and the corresponding bit in string I2 are different, place a 1 in the corresponding location in the output bit string.	8-5
NOT	Logical Invert	Set the state of each bit in output bit string Q to the opposite state of the corresponding bit in bit string I1.	8-7
SHL	Shift Left	Shift all the bits in a word or string of words to the left by a specified number of places.	8-9
SHR	Shift Right	Shift all the bits in a word or string of words to the right by a specified number of places.	8-9
ROL	Rotate Left	Rotate all the bits in a string a specified number of places to the left.	8-12
ROR	Rotate Right	Rotate all the bits in a string a specified number of places to the right.	8-12
BTST	Bit Test	Test a bit within a bit string to determine whether that bit is currently 1 or 0.	8-14
BSET	Bit Set	Set a bit in a bit string to 1.	8-16
BCLR	Bit Clear	Clear a bit within a string by setting that bit to 0.	8-16
BPOS	Bit Position	Locate a bit set to 1 in a bit string.	8-18
MCMP	Masked Compare	Compare the bits in the first string with the corresponding bits in the second.	8-20

Note

Note that, for all bit operations, the bit group of function blocks not explicitly bit-typed will affect the transitions (coils and contacts) for all bits in the written byte/word/dword. Please read the second example shown on page for further explanation.

AND and OR   (WORD, DWORD)

Each scan that power is received, the AND or OR function examines each bit in bit string I1 and the corresponding bit in bit string I2, beginning at the least significant bit in each.

For each two bits examined for the AND function, if both bits are 1, then a 1 is placed in the corresponding location in output string Q. If either or both bits are 0, then a 0 is placed in string Q in that location.

The AND function is useful for building masks or screens, where only certain bits are passed through (those that are opposite a 1 in the mask), and all other bits are set to 0. The function can also be used to clear the selected area of word memory by ANDing the bits with another bit string known to contain all 0s. The I1 and I2 bit strings specified may overlap.

For each two bits examined for the OR function, if either or both bits are 1, then a 1 is placed in the corresponding location in output string Q. If both bits are 0, then a 0 is placed in string Q in that location.

The OR function is useful for combining strings, and to control many outputs through the use of one simple logical structure. The function is the equivalent of two relay contacts in parallel multiplied by the number of bits in the string. It can be used to drive indicator lamps directly from input states, or superimpose blinking conditions on status lights.

The string length can be up to 256 words or double words for either function.

The function passes power flow to the right whenever power is received.
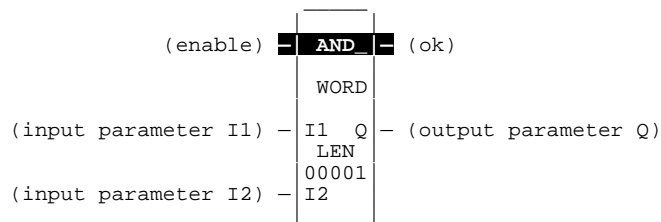
```
                              _____
         (enable) ─ | AND_ | ─ (ok)
                         | WORD |
 (input parameter I1) ─ |I1   Q| ─ (output parameter Q)
                         | LEN  |
                         |00001 |
 (input parameter I2) ─ |I2    |
                         |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first word.
I2	I2 contains a constant or reference for the second word.
ok	The ok output is energized whenever enable is energized.
Q	Output Q contains the result of the operation.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
I2	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o	o†	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00001 is set, the 16-bit strings represented by nicknames WORD1 and WORD2 are examined. The results of the logical AND are placed in output string RESULT.

```
 %I00001 |‾‾‾‾‾|
—| |——|  AND_ |—
         | WORD |
         |      |
 WORD1 —|I1   Q|—RESULT
         | LEN  |
         |00001 |
 WORD2 —|I2    |
         |_____|
```

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

XOR   (WORD, DWORD)

The Exclusive OR (XOR) function is used to compare each bit in bit string I1 with the corresponding bit in string I2. If the bits are different, a 1 is placed in the corresponding position in the output bit string.

Each scan that power is received, the function examines each bit in string I1 and the corresponding bit in string I2, beginning at the least significant bit in each. For each two bits examined, if only one is 1, then a 1 is placed in the corresponding location in bit string Q. The bit string length can be up to 256 words or double words. The XOR function passes power flow to the right whenever power is received.

If string I2 and output string Q begin at the same reference, a 1 placed in string I1 will cause the corresponding bit in string I2 to alternate between 0 and 1, changing state with each scan as long as power is received. Longer cycles can be programmed 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).

The XOR function is useful for quickly comparing two bit strings, or to blink a group of bits at the rate of one ON state per two scans.
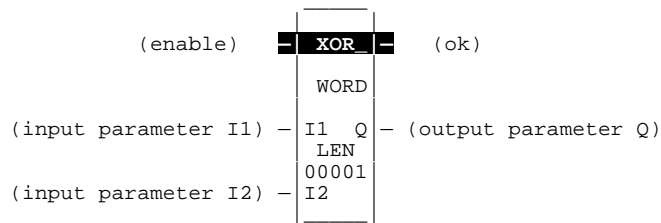
```
                                    ┌─────┐
          (enable)            ─┤ XOR_ ├─    (ok)
                                    │     │
                                    │ WORD│
 (input parameter I1) ─┤I1   Q├─ (output parameter Q)
                                    │ LEN │
                                    │00001│
 (input parameter I2) ─┤I2   │
                                    └─────┘
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first word to be XORed.
I2	I2 contains a constant or reference for the second word to be XORed.
ok	The ok output is energized whenever enable is energized.
Q	Output Q contains the result of I1 XORed with I2.

Valid Memory Types:

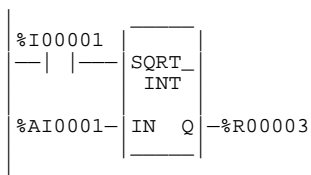| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
I2	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o	o†	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00001 is set, the 16-bit string represented by the nickname WORD3 is cleared (set to all zeros).

```
 |            _____
 |%I00001    |     |
 |—| |———|  XOR_ |—
 |           | WORD|
 |           |     |
 | WORD3 —|I1  Q|—WORD3
 |           | LEN |
 |           |00001|
 | WORD3 —|I2   |
 |           |_____|
 |
```
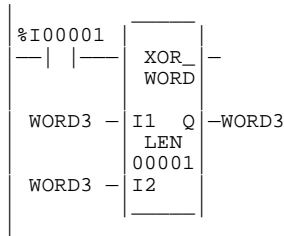
I1 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0
I2 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0

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

NOT (WORD, DWORD)

The NOT function is used to set the state of each bit in the output bit string Q to the opposite of the state of the corresponding bit in bit string I1.

All bits are changed on each scan that power is received, making output string Q the logical complement of I1. A length of 1 to 256 words or double words can be selected. The function passes power flow to the right whenever power is received.

```
                          _____
   (enable)        ─| NOT_ |─  (ok)
                         |      |
                         | WORD |
(input parameter I1) ─|IN   Q|─  (output parameter Q)
                         | LEN  |
                         |00001 |
                         |_____|
```
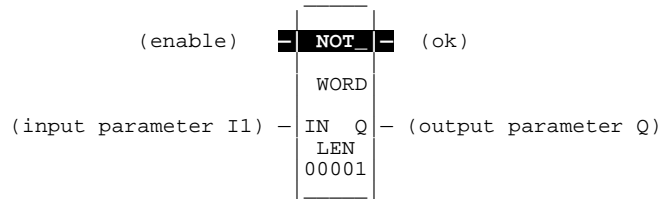
Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the word to be negated.
ok	The ok output is energized whenever enable is energized.
Q	Output Q contains the NOT (negation) of I1.

Valid Memory Types:

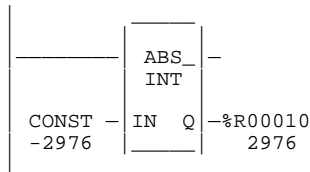| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o	o†	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00001 is set, the bit string represented by the nickname TAC is set to the inverse of bit string CAT.

```
|        ┌─────┐
|%I00001 |     |
|─| |────| NOT_|─
|        | WORD|
|        |     |
|   CAT ─|IN  Q|─TAC
|        | LEN |
|        |00001|
|        └─────┘
|
```
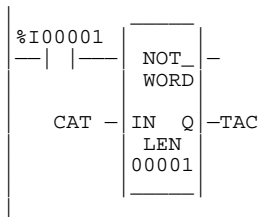
SHL and SHR   (WORD, DWORD)

The Shift Left (SHL) function is used to shift all the bits in a word or group of words to the left by a specified number of places. 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, the same number of bits is shifted in at the low end.

```
           MSB                                           LSB
B2 ←  | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |  ← B1
```
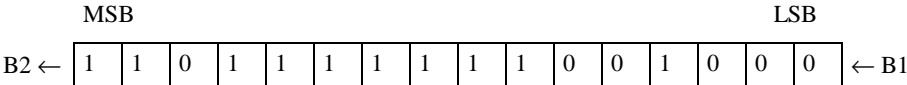
The Shift Right (SHR) function is used to shift all the bits in a word or group of words a specified number of places to the right. When the shift occurs, the specified number of bits is shifted out of the output string to the right. As bits are shifted out of the low end of the string, the same number of bits is shifted in at the high end.
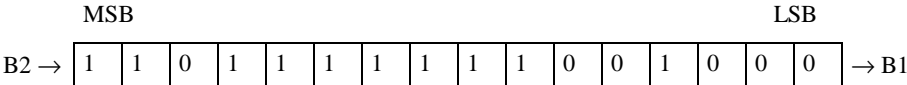
```
           MSB                                           LSB
B2 →  | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |  → B1
```

A string length of 1 to 256 words or double words can be selected for either function.

The number of places specified for the shift must be more than zero and less than the number of bits in the string. Otherwise, no shift occurs and no power flow is generated.

The bits being shifted into the beginning of the string are specified via input parameter B1. If a length greater than 1 has been specified as the number of bits to be shifted, each of the bits 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 special reference nickname ALW_ON as a permissive to input B1.

- All 0s. To do this, use the special reference nickname ALW_OFF as a permissive to input B1.

The SHL or SHR function passes power flow to the right, unless the number of bits specified to be shifted is greater than the total length of the string or is zero.

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.

```
                           ┌─────┐
        (enable)      ─┤ SHL_ ├─    (ok)
                       │     │
                       │ WORD │
(word to be shifted) ─┤ IN B2├─ (last bit shifted out)
                       │ LEN  │
                       │ 00001│
       (number of bits) ─┤N   Q├─ (output parameter Q)
                       │     │
        (bit shifted in) ─┤B1   │
                           └─────┘
```
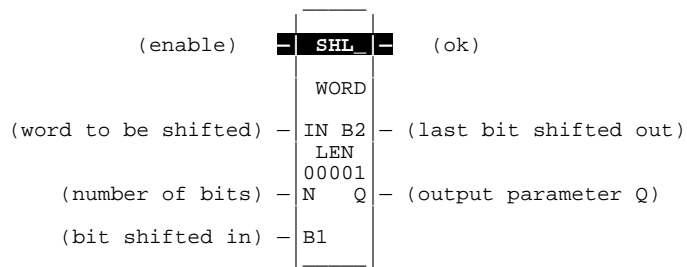
Parameters:

Parameter	Description
enable	When the function is enabled, the shift is performed.
IN	IN contains the first word to be shifted.
N	N contains the number of places (bits) that the array is to be shifted.
B1	B1 contains the bit value to be shifted into the array.
B2	B2 contains the bit value of the last bit shifted out of the array.
ok	The ok output is energized when the shift is energized and the shift length is not greater than the array size.
Q	Output Q contains the first word of the shifted array.
LEN	LEN is the number of words in the array to be shifted.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
N	•	•	•	•	•		•		•	•	•	•	•	•	•	
B1	•															
B2	•															•
ok	•															•
Q	•	o	o	o	o	o†	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00001 is set, the output bit string represented by the nickname WORD2 is made a copy of WORD1, left-shifted by the number of bits represented by the nickname LENGTH. The resulting open bits at the beginning of the output string are set to the value of %I00002.

```
|          _____
|%I00001  |       |
|—| |——|  SHL_ |—
|         | WORD |
|         |       |
| WORD1 —| IN B2 |—OUTBIT
|         | LEN   |
|         | 00001 |
|LENGTH —|N    Q |—WORD2
|    8    |       |
|%I00002  |       |
|—| |——| B1    |
|         |_____|
```
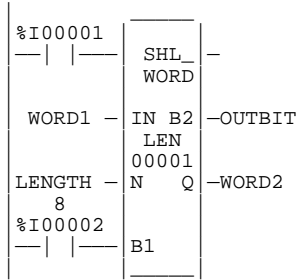
ROL and ROR   (WORD, DWORD)

The Rotate Left (ROL) function is used to rotate all the bits in a string a specified number of places to the left.   When rotation occurs, the specified number of bits is rotated out of the input string to the left and back into the string on the right.

The Rotate Right (ROR) function rotates the bits in the string to the right. When rotation occurs, the specified number of bits is rotated out of the input string to the right and back into the string on the left.

A string length of 1 to 256 words or double words can be selected for either function.

The number of places specified for rotation must be more than zero and less than the number of bits in the string. Otherwise, no movement occurs and no power flow is generated.

The ROL or ROR function passes power flow to the right, unless the number of bits specified to be rotated is greater than or equal to the total length of the string or is less than or equal to zero.

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.

```
                              _____
                             |     |
        (enable)       ─|   ROL_  |─    (ok)
                             |       |
                             | WORD  |
(word to be rotated) ─| IN   Q |─ (output parameter Q)
                             | LEN   |
                             | 00001 |
      (number of bits) ─| N     |
                             |_____|
```
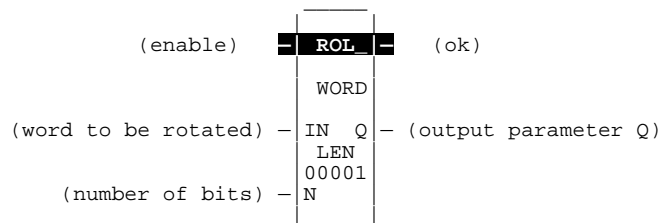
Parameters:

Parameter	Description
enable	When the function is enabled, the rotation is performed.
IN	IN contains the first word to be rotated.
N	N contains the number of places that the array is to be rotated.
ok	The ok output is energized when the rotation is energized and the rotation length is not greater than the array size.
Q	Output Q contains the first word of the rotated array.
LEN	LEN is the number of words in the array to be rotated.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
N	•	•	•	•	•		•		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o	o†	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00001 is set, the input bit string %R00001 is rotated 3 bits and the result is placed in %R00002. After execution of this function, the input bit string %R00001 is unchanged. If the same reference is used for IN and Q, a rotation will occur in place.

```
 |      _____
 |%I00001|     |
 |—| |——| ROL_ |—
 |       | WORD|
 |       |     |
 |%R00001—|IN  Q|—%R00002
 |       | LEN |
 |       |00001|
 | CONST —|N    |
 | +00003 |_____|
 |
```

%R00001:

MSB   LSB

← | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ←

%R00002 (after %I00001 is set):

MSB   LSB

| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

BTST   (WORD, DWORD)

The Bit Test (BTST) function is used to test 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 sweep power is received, the BTST 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 block can test different bits on successive sweeps. If the value of BIT is outside the range $(1 \le BIT \ge (16 * LEN))$, then Q is set OFF.

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

```
                          _____
(enable)          ─| BIT_ |─      (ok)
                         |
                    TEST_|
                     WORD|
(bit to be tested) ─|IN  Q|─ (output parameter Q)
                     LEN |
                    00001|
(bit number of IN) ─|BIT |
                     _____
```
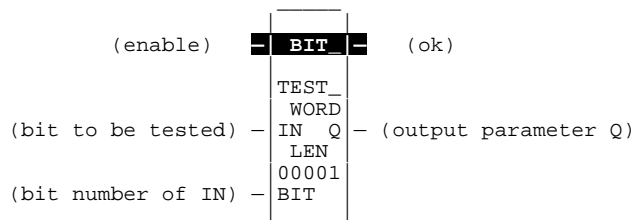
Parameters:

Parameter	Description
enable	When the function is enabled, the bit test is performed.
IN	IN contains the first word of the data to be operated on.
BIT	BIT contains the bit number of IN that should be tested. Valid range is $(1 \le BIT \ge (16 * LEN))$.
ok	The ok output is energized when enable is energized and BIT is greater than the string length or is zero.
Q	Output Q is energized if the bit tested was a 1.
LEN	LEN is the number of words in the string to be tested.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o	o	o		•	•	•	•	•	•	•	
BIT	•	•	•	•	•		•		•	•	•	•	•	•	•	
ok	•															•
Q	•															

<u>Note:</u> Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD..

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00001 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 I1.

```
|         _____
|%I00001 |      |
|—| |——— | BIT_ |—
|        |TEST_ |
|        |WORD  |                _____
|PRD_CDE—|IN  Q |————————————   |      |
|        |LEN   |               | ADD_ |—
|        |00001 |               | UINT |
|PICKBIT—|BIT   |       FND_ON— |I1  Q |—FND_ON
|        |_____|               |      |
|                               |      |
|                       CONST — |I2    |
|                       00001   |_____|
|
```
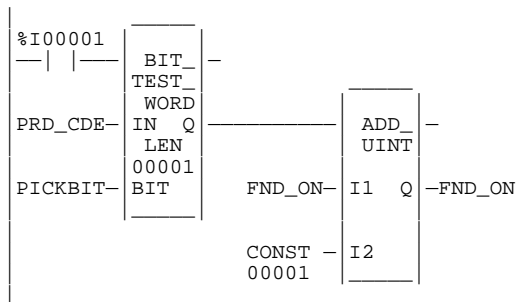
BSET and BCLR   (WORD, DWORD)

The Bit Set (BSET) function is used to set a bit in a bit string to 1. The Bit Clear (BCLR) function is used to clear a bit within a string by setting that bit to 0.

Each sweep that power is received, the function sets the specified bit to 1 for the BSET function or to 0 for the BCLR function. If a variable (register) rather than a constant is used to specify the bit number, the same function block can set different bits on successive sweeps.

A string length of 1 to 256 words or double words can be selected. The function passes power flow to the right, unless the value for BIT is outside the range (1 _ BIT _ (16 * LEN)). Then, ok is set OFF.

```
                         _____
                        |      |
        (enable)   ─█ BIT_ █─   (ok)
                        |      |
                        | SET_ |
                        | WORD |
    (first word) ─|IN    |
                        | LEN  |
                        |00001 |
(bit number of IN) ─|BIT   |
                        |_____|
```
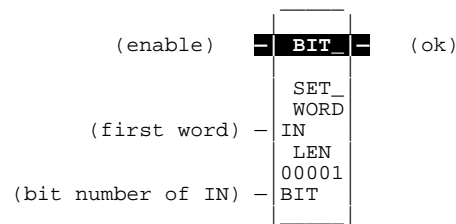
Parameters:

Parameter	Description
enable	When the function is enabled, the bit operation is performed.
IN	IN contains the first word of the data to be operated on.
BIT	BIT contains the bit number of IN that should be set or cleared. Valid range is (1≤BIT≥(16 * LEN)).
ok	The ok output is energized when enable is energized.
LEN	LEN is the number of words in the bit string.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN		o	o	o	o	o†	o		•	•	•	•	•	•		
BIT	•	•	•	•	•		•		•	•	•	•	•	•	•	
ok	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2. Also, please note that, for all bit operations, the bit group of function blocks not explicitly bit-typed will affect the transitions (coils and contacts) for all bits in the written byte/word/dword. Please read the second example shown below and the explanation that precedes it before using bit operations with byte and word blocks that have transitions.

Example 1:

In the following example, whenever input %I00001 is set, bit 12 of the string beginning at reference %L00040 is set to 1.

```
 |         _____
 |%I00001 |      |
 —| |—— | BIT_ |—
 |        | SET_ |
 |        | WORD |
 |%L00040—| IN   |
 |        | LEN  |
 |        | 00001|
 | CONST —| BIT  |
 | 00012  |_____|
 |
```
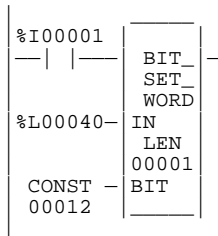
Example 2:

In the following example, M41–M48 will be solved as written to be a Transition Status. **These bits may not perform as expected** when used as a transition contact or coil. If you wish to use Bit Op functions *in conjunction with transition functions*, your Bit Op function should be type BIT.

```
 |         _____
 |%I00001 |      |
 —| |—— | BIT_ |—
 |        | SET_ |
 |        | WORD |
 |%M00041—| IN   |
 |        | LEN  |
 |        | 00001|
 | CONST —| BIT  |
 | 00003  |_____|
 |
```
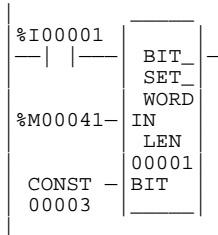
BPOS   (WORD, DWORD)

The Bit Position (BPOS) function is used to locate a bit set to 1 in a bit string.

Each sweep 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 can be selected. The function passes power flow to the right whenever enable is ON.

```
                       _____
                      |      |
  (enable)    ─|  BIT_ |─    (ok)
                      | POS_ |
                      | WORD |
(first word) ─| IN  Q |─
                      | LEN  |
                      |00001 |
                      |  POS |─ (position of non_zero bit or 0)
                      |_____|
```
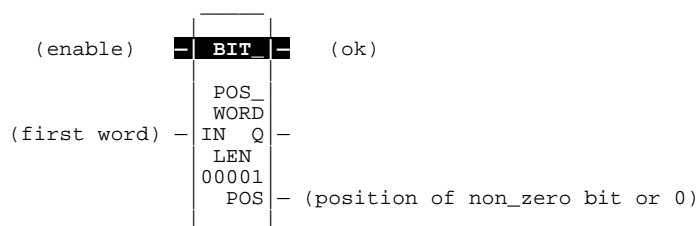
Parameters:

Parameter	Description
enable	When the function is enabled, a bit search operation is performed.
IN	IN contains the first word of the data to be operated on.
ok	The ok output is energized when enable is energized.
POS	The position of the first non-zero bit found, or zero if a non-zero bit is not found.
Q	Output Q is energized if a bit set to 1 is found.
LEN	LEN is the number of words in the bit string.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o	o	o		•	•	•	•	•	•	•	•
POS	•	•	•	•	•		•		•	•	•	•	•	•		
ok	•															•
Q	•															•

<u>Note</u>: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, if %I00001 is set, the bit string starting at %M00001 is searched until a bit equal to 1 is found, or 6 words have been searched. Coil %Q00001 is turned on. If a bit equal to 1 is found, its location within the bit string is written to %AQ0001 and %Q00002 is turned on. If %I00001 is set, bit %M00001 is 0, and bit %M00002 is 1, then the value written to %AQ0001 is 2.

```
|     _____
|%I00001 |      |                                                   %Q00001
|—| |——| BIT_ |———————————————————————————————————————————————————( )—
|        | POS_ |
|        | WORD |                                                   %Q00002
|%M00001—|IN  Q |———————————————————————————————————————————————————( )—
|        | LEN  |
|        |00006 |
|        |  POS |—%AQ0001
|        |_____|
|
```
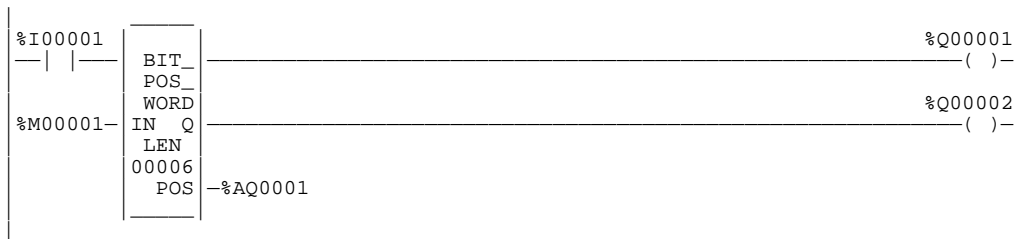
MCMP   (WORD, DWORD)

The Masked Compare (MCMP) function is used to compare the contents of two bit strings. Input string I1 might contain the states of outputs, such as solenoids or motor starters. Input string I2 might contain their input state feedback, such as limit switches or contacts.

Each scan that power is received, the function begins comparing the bits in the first string with the corresponding bits in the second. Comparison continues until a miscompare is found, or until the end of the string is reached.

The BIT input is used to indicate 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).

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 reset to the beginning of the input string before starting the next comparison.

The function passes power flow whenever power is received. The other outputs of the function depend on the state of the corresponding mask bit, as described below.

If All Bits in I1 and I2 are the Same

If all corresponding bits in strings I1 and I2 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 MCMP, it will be reset to 1.

If a Miscompare is Found

When the two bits currently being compared are not the same, the function then checks the correspondingly-numbered bit in string M (the mask). If the mask bit is a 1, the comparison continues until another miscompare or the end of the input strings is reached.

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

1. Sets the corresponding mask bit to a 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 (for example, 6).

5. Stops the comparison.

```
                      _____
     (enable)     ─■│MASK_│■─    (ok)
                    │      │
                    │COMP_ │
                    │ WORD │
(input parameter I1) ─│I1 MC│─ (miscompare)
                    │ LEN  │
                    │00001 │
(input parameter I2) ─│I2  Q│─ (output parameter Q)
                    │      │
    (bit string mask) ─│M  BN│─ (bit number for last miscompare)
                    │      │
      (bit number) ─│BIT   │
                    │_____│
```
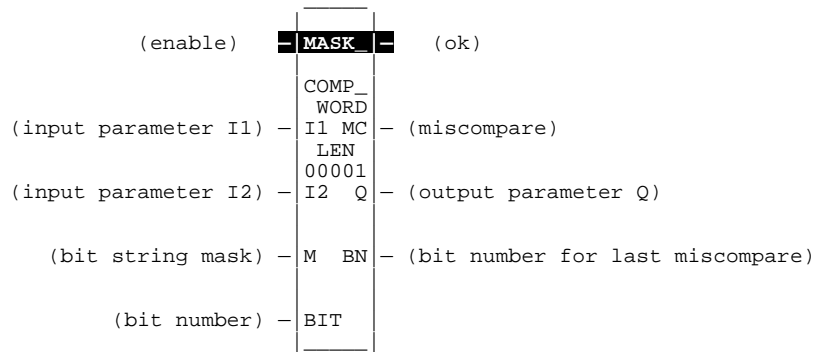
Parameters:

Parameter	Description
enable	Permissive logic to enable the function.
I1	Reference for the first bit string to be compared.
I2	Reference for the second bit string to be compared.
M	Reference for the bit string mask.
BIT	Reference which indicates where the next comparison should start (may be a constant value). The comparison begins 1 bit after BIT. For detailed discussion of this, please read the third paragraph on the preceding page.
ok	The ok output is energized when enable is energized.
MC	User logic to determine if a miscompare has occurred.
Q	Output copy of the mask (M) bit string.
BN	Number of the bit where the latest miscompare occurred.
LEN	LEN is the number of words in the bit string.

Valid Memory Types:

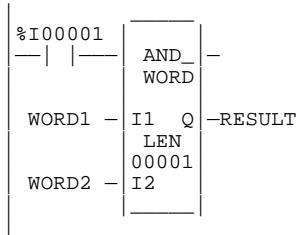| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
I1	•	o	o	o	o	o	o		•	•	•	•	•	•		
I2	•	o	o	o	o	o	o		•	•	•	•	•	•		
M		o	o	o	o	o†	o		•	•	•	•	•	•		
BIT	•	•	•	•	•		•		•	•	•	•	•	•	•	
ok	•															•
MC	•															•
Q		o	o	o	o	o†	o		•	•	•	•	•	•		
BN		•	•	•	•		•		•	•	•	•	•	•		

<u>Note</u>: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever %I00001 is set, the function compares the bits represented by the reference VALUES against the bits represented by the reference EXPECT. Comparison begins at the bit number specified in BITNUM. If an unmasked miscompare is detected, the comparison stops. The corresponding bit is set in the mask RESULT. 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 the function receives power flow.

```
        |    _____
 %I00001 |   |     |                                             %Q00001
 —| |——  |—| MASK_ |————————————————————————————————————————————( )—
            | COMP_ |
            | WORD  |                                             %Q00002
  VALUES—   | I1 MC |————————————————————————————————————————————( )—
            | LEN   |
            | 00025 |
  EXPECT—   | I2  Q |—NEWVALS
            |       |
            |       |
  RESULT—   | M  BN |—BITNUM
            |       |
            |       |
  BITNUM—   | BIT   |
            |_____|
```
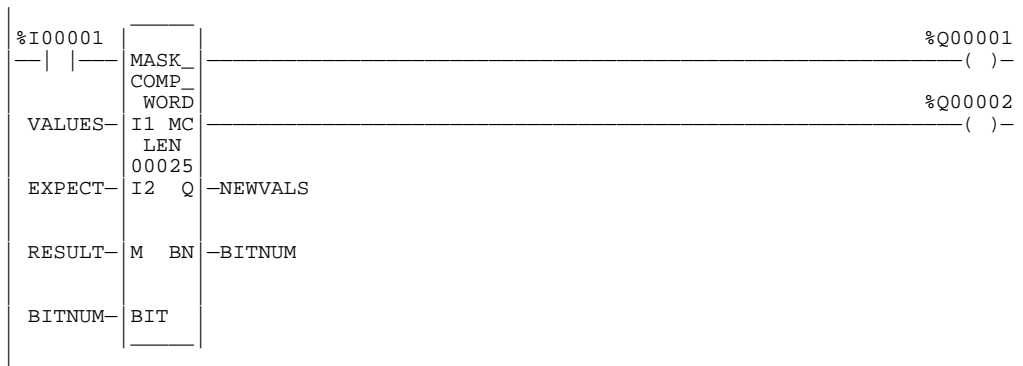
Chapter 9

Data Move Functions

Data move functions provide basic data move capabilities. This chapter describes the following data move functions:

Abbreviation	Function	Description	Page
MOVE	Move	Copy data as individual bits. The maximum length allowed is 32,767, except for MOVE_BIT which is 256 bits. Data can be moved into a different data type without prior conversion.	9-2
BLKMOV	Block Move	Copy a block of seven constants to a specified memory location. The constants are input as part of the function.	9-4
BLKCLR	Block Clear	Replace the content of a block of data with all zeros. This function can be used to clear an area of bit (%I, %Q, %M, %G, or %T) or word (%R, %P, %L, %AI, or %AQ) memory. The maximum length allowed is 256 words.	9-6
SHFR	Shift Register	Shift one or more data words into a table. The maximum length allowed is 256 words.	9-8
BITSEQ	Bit Sequencer	Perform a bit sequence shift through an array of bits. The maximum length allowed is 256 words.	9-11
SWAP	Swap	Swap two bytes of data within a word, or two words within a double word. The maximum length allowed is 256 words.	9-15
COMMREQ	Communications Request	Allow the program to communicate with an intelligent module, such as a Bus Controller, Programmable Coprocessor Module, or Subnet Module.	9-17
VMERD	VME Read	Read data from the VME backplane. The maximum length allowed is 32,767.	9-24
VMEWRT	VME Write	Write data to the VME backplane. The maximum length allowed is 32,767.	9-26
VMERMW	VME Read/Modify/Write	Update a data element using the read/modify/write cycle on the VME bus.	9-28
VMETST	VME Test and Set	Handle semaphores on the VME bus.	9-30
VME_CFG_RD	VME Configuration Read	Read the configuration for a VME module.	9-33
VME_CFG_WRT	VME Configuration Write	Write the configuration to a VME module.	9-36
DATA_INIT	Data Initialization	Copy a block of constant data to a reference range.	9-39
DATA_INIT_COMM	Data Initialize Communications Request	Initialize a COMMREQ function with a block of constant data. The length should equal the size of the COMMREQ function's entire command block.	9-42
DATA_INIT_ASCII	Data Initialize ASCII	Copy a block of constant ASCII text to a reference range. The length must be an even number.	9-45

MOVE   (INT, UINT, DINT, BIT, WORD, DWORD, REAL)

Use the MOVE function to copy data (as individual bits) from one location 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 location.

The MOVE function has two input parameters and two output parameters. When the function receives power flow, it copies data from the input parameter IN to the output parameter Q as bits. If data is moved from one location in discrete memory to another, (for example, 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. Data at the input parameter does not change unless there is an overlap in the source destination.

Input IN can be either a reference for the data to be moved or a constant. If a constant is specified, 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, the constant is placed in the memory location specified by Q and the locations following, up to the length specified. For example, if the constant value 9 is specified for IN and the length is 4, then 9 is placed in the memory location specified by Q and the three locations following.

The LEN operand specifies the number of:

- Words to be moved for MOVE_INT, MOVE_UINT, and MOVE_WORD.

- Double words to be moved for MOVE_DINT and MOVE_DWORD.

- Bits to be moved for MOVE_BIT.

- Reals to be moved for MOVE_REAL.

The function passes power to the right whenever power is received.

```
                              _____
                             |        |
           (enable)        ──| MOVE_ |──    (ok)
                             |        |
                             |  WORD  |
(value to be moved) ──|IN   Q|── (output parameter Q)
                             |  LEN   |
                             | 00001  |
                             |_____|
```
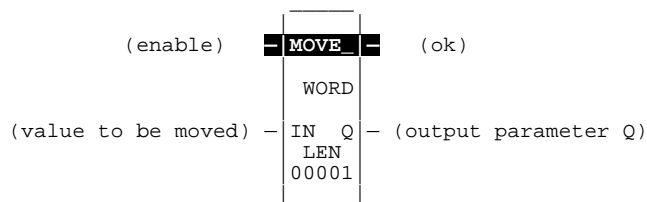
Parameters:

Parameter	Description
enable	When the function is enabled, the move is performed.
IN	IN contains the value to be moved. For MOVE_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 1 bit, beginning with the reference address specified, is displayed online.
ok	The ok output is energized whenever the function is enabled.
Q	When the move is performed, the value at IN is written to Q. For MOVE_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 1 bit, beginning with the reference address specified, is displayed online.
LEN	LEN must be between 1 and 32,767, except for MOVE_BIT, which is between 1 and 256 bits, unless IN is a constant. For MOVE_BIT, when IN is a constant, LEN must be between 1 and 16. The transition references of all bytes in the range are modified.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o	Δ	o	o	•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o	†	o	o	•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.
o Valid reference for INT, UINT, BIT, or WORD data only; not valid for DINT, DWORD, or REAL.
For MOVE_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.
%U is allowed for MOVE_BIT only.
Δ Valid reference for BIT or WORD data only; not valid for INT, UINT, DINT, DWORD, or REAL.
† %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In this example, whenever %I00003 is set, the three bits %M00001, %M00002, and %M00003 are moved to %M00100, %M00101, and %M00102, respectively. Coil %Q00001 is turned on.

```
|     _____
|%I00003  |      |                                                    %Q00001
|—| |——|MOVE_ |————————————————————————————————————————————————————————( )—
|      |  BIT |
|      |      |
|%M00001—|IN  Q|—%M00100
|      | LEN  |
|      | 00003|
|      |_____|
|
```

BLKMOV   (INT, UINT, DINT, WORD, DWORD, REAL)

Use the Block Move (BLKMOV) function to copy a block of seven constants to a specified location.

The BLKMOV function has eight input parameters and two output parameters. When the function receives power flow, it copies the constant values into consecutive locations, beginning at the destination specified in output Q. Output Q cannot be the input of another program function.

Note

For BLKMOV_INT, the values of IN1 x IN 7 are displayed as signed decimals.
For BLKMOV_WORD, IN1 x IN7 are displayed in hexadecimal.

The function passes power to the right whenever power is received.

```
                              _____
        (enable)       ─| BLKMV |─    (ok)

                              WORD

  (constant value) ─| IN1 Q|─ (output parameter Q)

  (constant value) ─| IN2

  (constant value) ─| IN3

  (constant value) ─| IN4

  (constant value) ─| IN5

  (constant value) ─| IN6

  (constant value) ─| IN7
                              _____
```
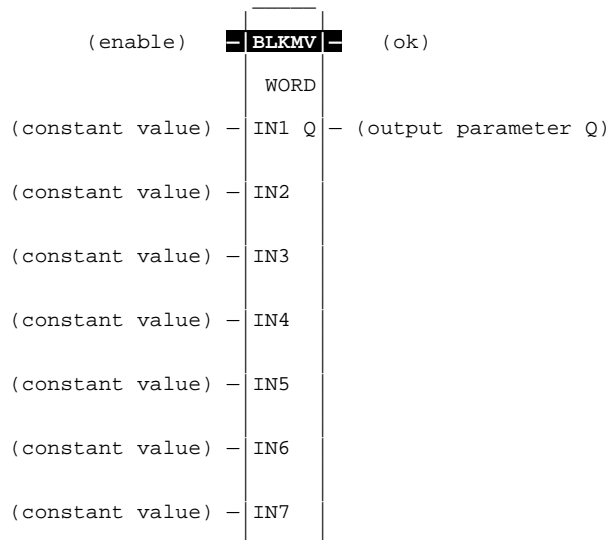
Parameters:

Parameter	Description
enable	When the function is enabled the block move is performed.
IN1 - IN7	IN1 through IN7 contain seven constant values.
ok	The ok output is energized whenever the function is enabled.
Q	Output Q contains the first integer of the moved array. IN1 is moved to Q.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN1-IN7															•	
ok	•															•
Q		o	o	o	o	Δ†	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, or WORD only; not valid for DINT, DWORD, or REAL.
Δ   Valid reference for WORD data only; not valid for INT, UINT, DINT, DWORD, or REAL.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, when the enabling input represented by the nickname FST_SCN is ON, the BLKMOV function copies the seven input constants into memory locations %L00010 through %L00016.

```
         _____
 FST_SCN |       |
 —| |—— | BLKMV |—
         |  INT  |
         |       |
 CONST — | IN1 Q |—%L00010
 +32767  |       |
         |       |
 CONST — | IN2   |
 _32768  |       |
         |       |
 CONST — | IN3   |
 +00001  |       |
         |       |
 CONST — | IN4   |
 +00002  |       |
         |       |
 CONST — | IN5   |
 +00002  |       |
         |       |
 CONST — | IN6   |
 +00001  |       |
         |       |
 CONST — | IN7   |
 +00001  |_____|
         |       |
```
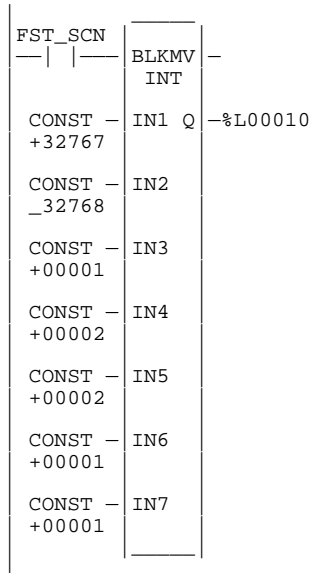
BLKCLR   (WORD)

Use the Block Clear (BLKCLR) function to fill a specified block of data with zeros.

The BLKCLR function has two input parameters and one output parameter. When the function receives power flow, it writes zeros into the memory location beginning at the reference specified by IN. When the data to be cleared is from discrete memory (%I, %Q, %M, %G, or %T), the transition information associated with the references is also cleared.

Note

The input parameter IN is not included in coil checking.

The function passes power to the right whenever power is received.

```
                                    _____
                                   |     |
         (enable)              ─| BLK_ |─    (ok)
                                   |     |
                                   | CLR_ |
                                   | WORD |
(word to be cleared)  ─| IN   |
                                   | LEN  |
                                   |00001|
                                   |_____|
```
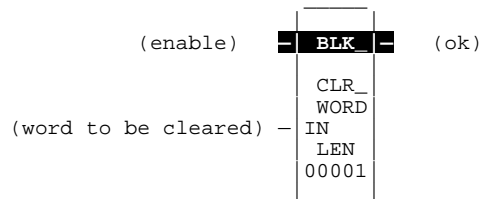
Parameters:

Parameter	Description
enable	When the function is enabled, the array is cleared.
IN	IN contains the first word of the array to be cleared.
ok	The ok output is energized whenever the function is enabled.
LEN	LEN must be between 1 and 256 words.

Valid Memory Types:

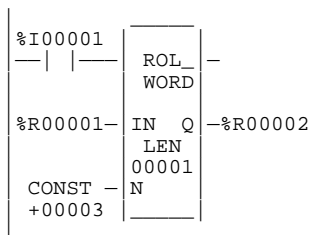| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN		•	•	•	•	•†	•		•	•	•	•	•	•		
ok	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
†   %SA, %SB, %SC only; %S cannot be used.

Note
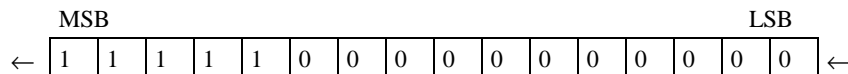
For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
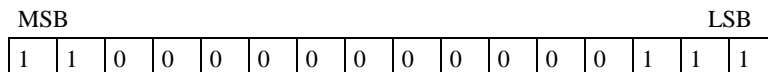
Example:

In the following example, at power-up, 32 words of %Q memory (512 points) beginning at %Q00001 are filled with zeros. The transition information associated with these references will also be cleared.

```
|          _____
| FST_SCN |      |
|—| |———  | BLK_ |—
|          | CLR_ |
|          | WORD |
| %Q00001—| IN    |
|          |  LEN  |
|          | 00032 |
|          |_____|
|
```
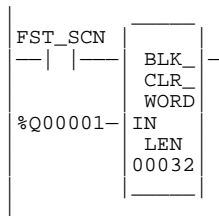
SHFR   (BIT, WORD, DWORD)

Use the Shift Register (SHFR) function to shift one or more data words or data bits from a reference location into a specified area of memory. 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.

Note

When assigning reference addresses, overlapping input and output reference address ranges in multi-word functions may produce unexpected results.

The SHFR function has five input parameters and two output parameters. 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 for LEN, are filled with zeros. LEN determines the length of the shift register.

If the function receives power flow and reset is not active, it shifts data in the shift register down by the number of elements (bit or word) specified in N. The last element in the shift register is shifted into Q. The rightmost element of IN is shifted into the vacated element starting at ST. 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 power is received through the enable logic.

```
                                    _____
                                   |      |
         (enable)            ─█| SHFR_ |█─     (ok)
                                   |      |
                                   | WORD |
                                   |      |
          (reset) ─| R      Q |─ (output parameter Q)
                                   | LEN  |
                                   |00001 |
 (number of elements) ─| N      |
                                   |      |
 (value to be shifted) ─| IN     |
                                   |      |
    (first bit or word) ─| ST     |
                                   |_____|
```
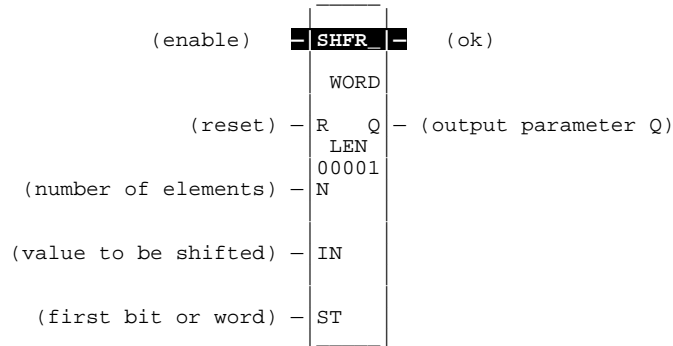
Parameters:

Parameter	Description
enable	When enable is energized and R is not, the shift is performed.
R	When R is energized, the shift register located at ST is filled with zeros.
N	N contains the number of elements to be shifted into the shift register.
IN	IN contains the value to be shifted into the first bit or word of the shift register. For SHFR_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 1 bit, beginning with the reference address specified, is displayed online.
ST	ST contains the first bit or word of the shift register. For SHFR_BIT, 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.
ok	The ok output is energized whenever the function is enabled.
Q	Output Q contains the bit or word shifted out of the shift register. For SHFR_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 1 bit, beginning with the reference address specified, is displayed online.
LEN	LEN must be between 1 and 256 bits, words, or double words.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
R	•															
N															•	
IN	*	o	o	o	o	o	o		•	•	•	•	•	•	•	
ST		o	o	o	o	o†	o		•	•	•	•	•	•		
ok	•															•
Q	*	o	o	o	o	o†	o	o	•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.
* Valid reference for WORD or DWORD data only; not valid for BIT.
o Valid reference for BIT or WORD data only; not valid for DWORD.
 For SHFR_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.
† %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example 1:

In the following example, the shift register operates on register memory locations %R00001 through %R00100. 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 active, the word from output status table location %Q00033 is shifted into the shift register. The word shifted out of the shift register is stored in output %M00005. Note that, for this example, the length specified for LEN and the amount of data to be shifted (N) are not the same.

```
|      _____
| NXT_CYC |      |
|—| |——| SHFR_ |—
|         | WORD |
| CLEAR   |      |
|—| |——| R   Q |—%M00005
|         | LEN  |
|         | 00100|
| CONST —| N    |
|  00001  |      |
|         |      |
|%Q00033—| IN   |
|         |      |
|         |      |
|%R00001—| ST   |
|         |_____|
|
```
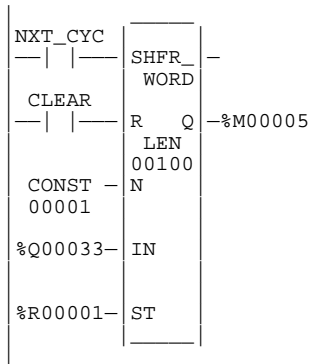
Example 2:

In this example, the shift register operates on memory locations %M00001 through %M00100. When the reset reference CLEAR is active, the SHFR function fills %M00001 through %M00100 with zeros.

When NXT_CYC is active and CLEAR is not, the SHFR function shifts the data in %M00001 to %M00100 down by one bit. The bit in %Q00033 is shifted into %M00001 while the bit shifted out of %M00100 is written to %M00200.

```
|      _____
| NXT_CYC |      |
|—| |——| SHFR_ |—
|         | BIT  |
| CLEAR   |      |
|—| |——| R   Q |—%M00200
|         | LEN  |
|         | 00100|
| CONST —| N    |
|  00001  |      |
|%Q00033—| IN   |
|         |      |
|         |      |
|%R00001—| ST   |
|         |_____|
|
```
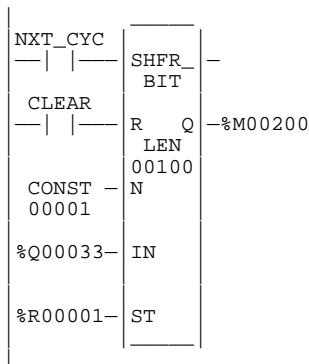
BITSEQ   (BIT)

The Bit Sequencer (BITSEQ) function performs a bit sequence shift through an array of bits. The BITSEQ function has five input parameters and one output parameter. The operation of the function depends on the previous value of the parameter EN, as shown in the following table.

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

The reset input (R) overrides the enable (EN) and always resets the sequencer. When R is active, the current step number is set to the value passed in via the step number parameter. If no step number is passed in, step is set to 1. All of the bits in the sequencer 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, the bit pointed to by the current step number is cleared. The current step number is either incremented or decremented, based on the direction parameter. 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 ($1 \leq$ step number $\geq$ LEN), it is set back to 1.

- When the step number is being decremented and it goes outside the range of ($1 \leq$ step number $\geq$ LEN), it is set to LEN.

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

Memory Required for a Bit Sequencer

Each bit sequencer uses three words (registers) of %R, %L, or %P memory to store the following information:

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

Note

Do **not** write to these registers from other functions.

When you enter a bit sequencer, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. For example:

```
                          _____
                         |     |
(enable)          ─ | BIT_|─   (ok)
                         |
                         |  SEQ
                         |
(reset)           ─ |R
                         |  LEN
                         | 00001
(direction)       ─ |DIR
                         |
                         |
(number)          ─ |STEP
                         |             this table has no title
                         |
(starting address) ─ |ST
                         |
                         |_____|

                          (address)
```
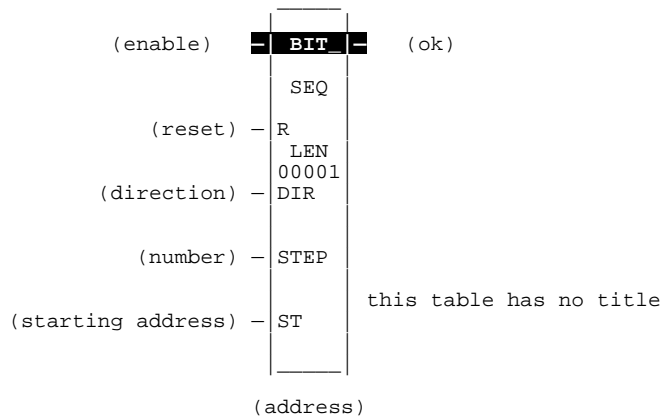
The control word stores the state of the Boolean inputs and outputs of its associated function block, as shown in the following format:

15	14	13	12	11	10	9	8

7	6	5	4	3	2	1	0
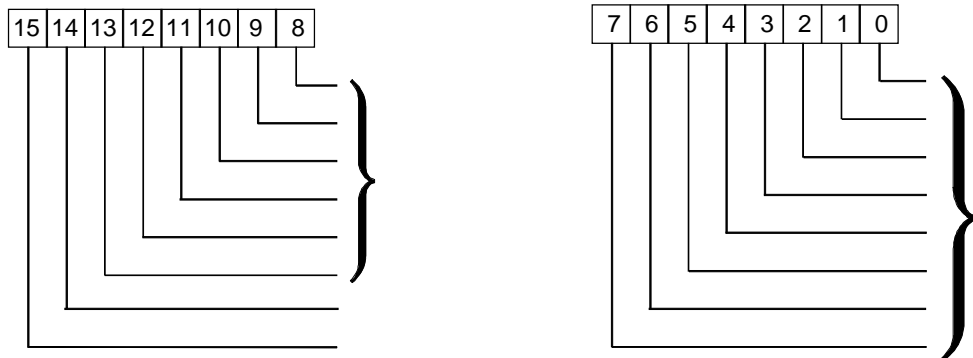
Note

Bits 0 through 13 are not used.

Parameters:

Parameter	Description
address	The bit sequencer uses three consecutive words (registers) of %R, %P, or %L memory to store the: • Current step number = word 1. • Length of sequence in bits = word 2. • Control word = word 3. When you enter a bit sequencer, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. For more information, refer to the preceding page. **Note**: Do not use this address with other instructions. **Caution**: Overlapping references will result in erratic operation of the bit sequencer.
enable	When the function is enabled, if it was not enabled on the previous sweep and if R is not energized, the bit sequence shift is performed.
R	When R is energized, the bit sequencer's step number is set to the value in STEP (default = 1), and the bit sequencer is filled with zeros, except for the current step number bit.
DIR	When DIR is energized, the bit sequencer's step number is incremented prior to the shift. Otherwise, it is decremented.
STEP	When R is energized, the step number is set to this value.
ST	ST contains the first word of the bit sequencer.
ok	The ok output is energized whenever the function is enabled.
LEN	LEN must be between 1 and 256 words.

Note

Coil checking for the BITSEQ function checks for 16 bits from the ST parameter, even when LEN is less than 16.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
address									•					•		
enable	•															
R	•															
DIR	•															
STEP		•	•	•	•		•		•	•	•	•	•	•	•	•
ST		•	•	•	•	†	•		•	•	•		•	•		•
ok	•															

• Valid reference or place where power may flow through the function.
† %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the sequencer operates on register memory %R00001. Its static data is stored in registers R00004, R00005, and R00006. When CLEAR is active, the sequencer is reset and the current step is set to step number 3. The first 8 bits of %R00001 are set to zero.

When NXT_SEQ 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 DIR is energized) is set.

```
 |         _____
 |NXT_SEQ |      |
 |—| |——— | BIT_ |—
 |        |  SEQ |
 | CLEAR  |      |
 |—| |——— |R     |
 |        |  LEN |
 | DIRECT |00008 |
 |—| |——— |DIR   |
 |        |      |
 |        |      |
 | CONST —|STEP  |
 | 00003  |      |
 |        |      |
 |%R00001—|ST    |
 |        |_____|
 |         %R00004
 |
```
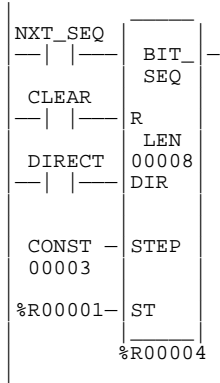
SWAP   (WORD, DWORD)

Use the SWAP function to swap two bytes within a word, or two words within a double word.

The SWAP function can be performed over a range of memory by specifying a length greater than 1 for the function. If this is done, each word or double word of data within the specified length is appropriately swapped.

The SWAP function has two input parameters and two output parameters. When the SWAP function receives power flow, it performs the swap operation on each word or double word of data within the specified area. The results of the swap are stored to output Q.

The SWAP function passes power to the right whenever power is received.

```
                             ┌─────┐
        (enable)        ─┤ SWAP_ ├─     (ok)
                         │ WORD  │
(input parameter IN) ─┤ IN  Q ├─ (output parameter Q)
                         │ LEN   │
                         │ 00001 │
                         └─────┘
```
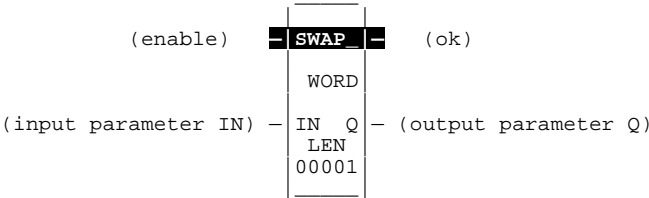
Parameters:

Parameter	Description
enable	When the function is enabled, the swap is performed.
IN	IN contains the data to be swapped.
ok	The ok output is energized when enable is energized.
Q	Output Q contains the swapped version of the original data IN.
LEN	LEN must be between 1 and 256 words.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o		o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for WORD data only; not valid for DWORD.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, two bytes located in bits %I00033 through %I00048 are swapped. The result is stored in %L00007.

```
|          _____
|         |     |
|_____|SWAP_|_
|         |WORD |
|         |     |
|%I00033__|IN  Q|__%L00007
|         |LEN  |
|         |00001|
|         |_____|
|
```

COMMREQ

Use the Communication Request (COMMREQ) function if the program needs to communicate with an intelligent module such as a Bus Controller, Programmable Coprocessor Module, or LAN Interface Module. The information presented on the following pages shows the format of the COMMREQ function. You will need additional information to program the COMMREQ for each type of device. Programming requirements for each type of module that uses the COMMREQ function are described in the module's documentation.

Note

If you are using Serial Communications, refer to the *Series 90™ PLC Serial Communications User's Manual* (GFK-0582). If you are using MMS-Ethernet Communications, refer to the *MMS-Ethernet Communications for the Series 90™-70 PLC User's Manual* (GFK-0686).

As an example of the types of communications that can be requested, the Genius Bus Controller uses the following types of messages:

Command	Description
1	Pulse test outputs.
2	Read configuration data from a device on the bus.
3	Write configuration data to a device on the bus.
4	Read diagnostics.
5	Clear circuit fault.
6	Clear all circuit faults.
7	Assign monitor.
8	Enable/disable outputs.
9	Enable/disable global data.
10	Switch Bus Switching Module.
11	Read device.
12	Write device.
13	Dequeue datagram.
14	Read datagram.
15	Receive datagram.

The Send Datagram and Receive Datagram commands (14 and 15 above) can be used to transmit several additional messages. Refer to the *Bus Controller Manual*, GFK-0398, for programming information on the bus controller.

Note that devices can also exchange global data with the PLC during the System Communications Window, which occurs automatically at the end of each sweep. Such communications do not require program assistance.

The COMMREQ function has four input parameters and two output parameters. When the COMMREQ function receives power flow, a command block of data is sent to the communications TASK. The command block begins at the reference specified using the parameter IN. The device to be communicated with is indicated by entering its rack and slot number for SYSID.

The COMMREQ may either send a message and wait for a reply, or send a message and continue without waiting for a reply. If the command block specifies that the program will not wait for a reply, the command block contents are sent to the receiving device and the program execution resumes immediately. (The timeout value is ignored.) This is referred to as **NOWAIT** mode.

If the command block specifies that the program will wait for a reply, the command block contents are sent to the receiving device and the CPU waits for a reply. The maximum length of time the PLC will wait for the device to respond is specified in the command block. If the device does not respond in that time, program execution resumes. This is referred to as **WAIT** mode.

The function passes power flow, unless the timeout period is exceeded, or if a 0 timeout period has been specified. The Function Faulted (FT) output may be set ON if:

1. The specified target module is not present or is faulted..

2. The specified task is not valid for the device.

3. The data length is 0.

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 additional information needed by the COMMREQ function.

The address of the command block is specified for the IN input to the COMMREQ function. This address may be in any word-oriented user reference (%R, %L, %P, %AI, or %AQ). The length of the command block depends on the specific command being sent.

The command block has the following structure:

Data Block Length	address
Wait/No Wait Flag	address + 1
Status Pointer Memory Type	address + 2
Status Pointer Offset	address + 3
Idle Timeout Value	address + 4
Maximum Communication Time	address + 5
Data Block	address + 6 to address + 133

Information required for the command block can be placed in the designated memory area using the MOV, BLKMOV, or DATA_INIT function block.

When entering information for the command block, refer to these definitions:

Data Block Length   The number of data words starting with th data at address +6 to the end of the command block, inclusive. The data block length ranges from 1 to 128 words. Each COMMREQ 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

Wait/No Wait Flag: This selects whether or not the program should wait for CCM communications to be completed.

For	Enter
No wait	0
Wait for replay	1

The flag bit is stored in the least significant bit (LSB) at address + 1. The rest of the word should be filled with zeros.

Note

Wait mode COMMREQs cannot be directed to serial ports 1 and 2 on release 7.0 or later CPX CPUs.

Status Pointer Memory Type: The two status pointer words specify a PLC memory location where the status word returned by the device will be written when the COMMREQ completes.

Status Pointer Memory Type	address + 2
Status Pointer Offset	address + 3

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 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
%AI	Analog input table	10
%AQ	Analog output table	12

* Numbers shown are decimal.

Note

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.

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. %R00001, for example, is at offset zero in the register table.

Idle Timeout Value:	The idle timeout value is the maximum time the PLC CPU waits for the device to acknowledge receipt of the COMMREQ. This value is ignored in **NOWAIT** mode. If **WAIT** mode is selected, address + 4 specifies the idle timeout period in 100-microsecond increments.
Maximum Communica-tion Time:	The value at address +5 specifies the maximum time the PLC CPU waits for the device to complete the COMMREQ. This time is also specified in 100-microsecond increments and is ignored in **NOWAIT** mode.

Data Block

The data block contains the parameters of the command. 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 (i.e., PCM, GBC, Communications) for specific COMMREQ command formats.

```
                                          _____
            (enable)  ─ | COMM_ | ─   (ok)
                        |  REQ  |
(first word of Command block) ─ | IN  FT | ─
            (rack/slot number) ─ | SYSID  |
                    (task ID) ─ | TASK   |
                        |_____|
```
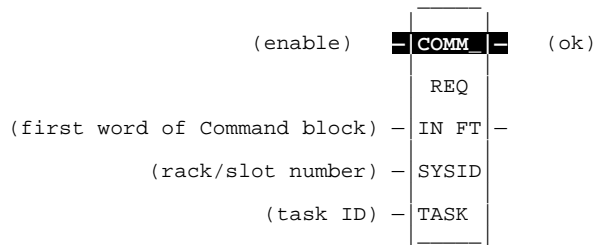
Parameters:

Parameter	Description
enable	When the function is energized, the communications request is performed.
IN	IN contains the first word of the command block.
SYSID	SYSID contains the rack number (most significant byte) and slot number (least significant byte) of the target device.
TASK	TASK contains the task ID of the process on the target device.
FT	FT is energized when the communication request fails. This was previously discussed in greater detail on page -92.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN									•	•	•	•	•	•		
SYSID		•	•	•	•		•		•	•	•	•	•	•	•	
TASK									•	•	•	•	•	•	•	
ok	•															•
FT	•															•

<u>Note</u>: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example 1:

In the following example, when enabling input %M00020 is ON, a command block located starting at %R00016 is sent to communications task 1 in the device located at rack 1, slot 2 of the PLC. If an error occurs, %Q00100 is set.

```
|       _____
|%M00020|       |
—| |——| COMM_ |—
|      | REQ   |
|      |       |                                          %Q00100
|%R00016| IN FT |——————————————————————————————————————————( )—
|      |       |
|      |       |
|CONST —| SYSID |
|0102  |       |
|      |       |
|CONST —| TASK  |
|00001 |_____|
|
```
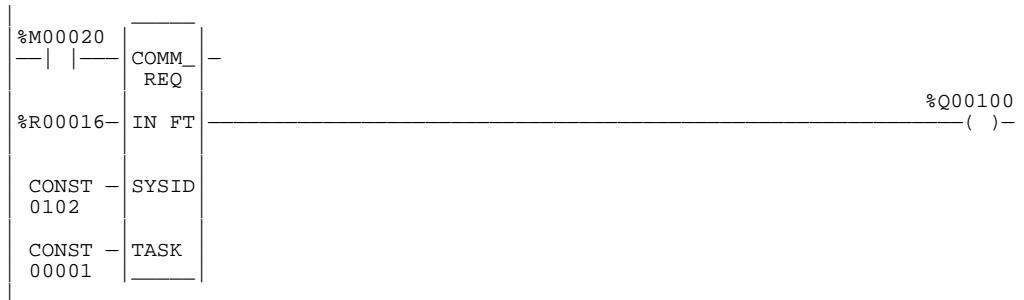
Example 2:

This example shows how the MOVE function can be used to enter command block contents for the COMMREQ described in example 1:

```
|
| FST_SCN                 _____                    _____                    _____
| —| |———————————————————| MOVE_ |———————————————————| MOVE_ |———————————————————| MOVE_ |
|                         | UINT  |                   | UINT  |                   | UINT  |
|                         |       |                   |       |                   |       |
|              CONST —| IN  Q |—%R00016   CONST —| IN  Q |—%R00017   CONST —| IN  Q |—%R00018
|              00100  | LEN   |           00001  | LEN   |           00008  | LEN   |
|                     | 00001 |                  | 00001 |                  | 00001 |
|                     |_____|                  |_____|                  |_____|
|
|                         _____                    _____                    _____
|              —————————————| MOVE_ |———————————————————| MOVE_ |———————————————————| MOVE_ |—
|                         | UINT  |                   | UNIT  |                   | UNIT  |
|                         |       |                   |       |                   |       |
|              CONST —| IN  Q |—%R00019   CONST —| IN  Q |—%R00020   CONST —| IN  Q |—%R00021
|              00512  | LEN   |           00100  | LEN   |           00200  | LEN   |
|                     | 00001 |                  | 00001 |                  | 00001 |
|                     |_____|                  |_____|                  |_____|
|                     _____
| %M00020            |       |
| —| |———————————————| COMM_ |—
|                    | REQ   |
|                    |       |                                                      %Q00100
| %R00016 —| IN FT |————————————————————————————————————————————————————————————————( )—
|                    |       |
|  CONST —| SYSID |
|  0102              |       |
|                    |       |
|  CONST —| TASK  |
|  00001             |_____|
|
```

Input IN of the COMMREQ specifies %R00016 as the beginning reference for the command block. Successive references contain the following:

Data Block Length	%R00016
Wait/No Wait Flag	%R00017
Status Pointer Memory Type	%R00018
Status Pointer Offset	%R00019
Idle Timeout Value	%R00020
Maximum Communication Time	%R00021
Data Block	%R00022 to end of data

MOVE functions supply the following command block data for the COMMREQ. The first MOVE function places the length of the data being communicated in %R00016. The second MOVE function places the constant 1 in %R00017. This specifies **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 next MOVE function places the constant 512 in reference %R00019. Therefore, the status pointer is located at %R00513. Additional MOVE functions place the constant 100 in %R00020 and 200 in %R00021. 100 is an idle timeout value of 10 milliseconds for the COMMREQ; 200 represents the maximum communications time of 20 milliseconds.

The programming logic displayed in example 2 on the previous page can be simplified by replacing the six MOVE functions with one DATA_INIT_COMM function.

```
|       _____
| FST_SCN |     |                                                          %Q00002 |——
|—| |——| DATA_|——————————————————————————————————————————————————————————————( )—
|       | INIT—|
|       | COMM |
|       |    Q |—%R00016
|       | LEN  |
|       | 00134|
|       |_____|
|
|
|       _____
| %M00020 |     |
|—| |——| COMM_|—
|       | REQ |
|       |     |                                                          %Q00100
| %R00016—| IN FT|——————————————————————————————————————————————————————————————( )—
|       |     |
|  CONST —| SYSID|
|  0012   |     |
|       |     |
|  CONST —| TASK |
|  00001  |_____|
|
```
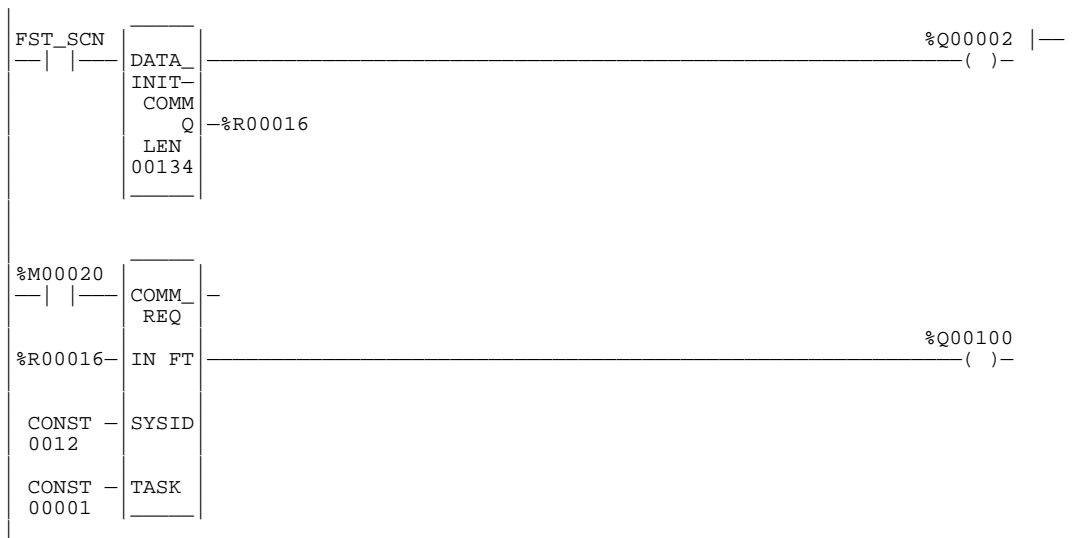
In Logicmaster, position the cursor on the DATA_INIT_COMM function block, and press **Zoom (F10)** to zoom into the DATA_INIT_COMM window.

VMERD   (BYTE, WORD)

The VME Read (VMERD) function is used to read data from the VME bus.

Note

Using a VME function (VMERD, VMEWRT, VMERMW, or VMETST) requires additional information on the correct way to address the VME board. This information may be obtained from one of two sources. For a qualified VME board, the VME board vendor may issue application notes on the correct use of the board. Otherwise, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

The VMERD function has three input parameters and one output parameter. When the VMERD function receives power flow, the function accesses the VME module at the address specified by ADR and the address modifier AM. It copies data with the length LEN to PLC locations beginning at output Q. The VMERD function passes power to the right when its operation is successful.

```
                            _____
    (enable)       _|  VME_ |_    (ok)
                    |  RD_   |
                    |  WORD  |
(address modifier) _| AM     |
                    |  LEN   |
                    | 00001  |
   (data address) _| ADR  Q |_  (output parameter Q)
                    |_____|
```
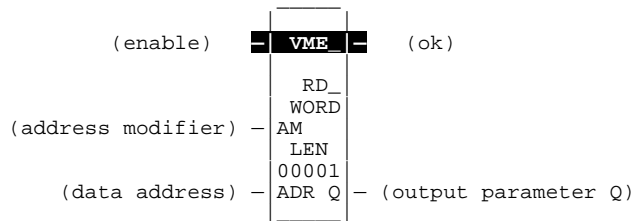
Parameters:

Parameter	Description
enable	When enable is energized, the VME read is performed.
AM	AM contains the address modifier.
ADR	ADR contains the address of the data to be read.
ok	The ok output is energized when the function is enabled and the data is successfully read.
Q	Output Q contains the data read from the address specified by ADR and AM.
LEN	LEN may be 1 to 32,767.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
AM	•								•	•	•	•	•	•	•	
ADR	•								•	•	•	•	•	•	•	
ok	•															•
Q	o	o	o	o	o		o		•	•	•	•	•	•		

<u>Note</u>:   Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for BYTE data only; not valid for WORD.

Note

> For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, when enabling input %I01001 goes ON, 256 bytes of short supervisory space are read from address 3800 on the VME bus into registers %R00001 through %R00128. (In a multiple rack system with a BTM, this would be on rack 4.) Unless an error occurs while reading the data, coil %Q00001 is set ON.

```
|         _____
| %I01001|     |                                                        %Q00001
|──| ↑ |──| VME_|────────────────────────────────────────────────────────( )─
|        | RD_ |
|        | BYTE|
| CONST ─|AM   |
|  0013  | LEN |
|        |00256|
| CONST ─|ADR Q|─%R00001
|00003800|_____|
|
```
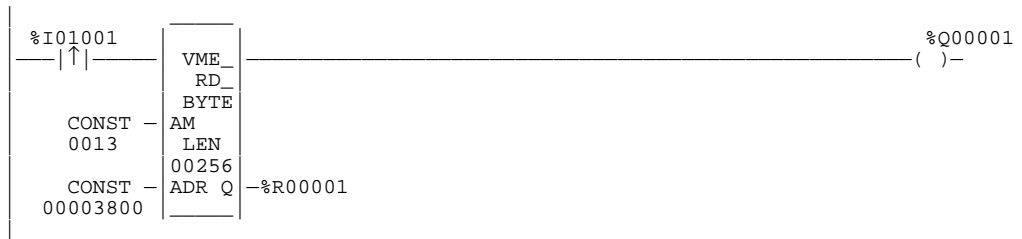
VMEWRT   (BYTE, WORD)

The VME Write (VMEWRT) function is used to write data to the VME bus.

Note

Using a VME function (VMERD, VMEWRT, VMERMW, or VMETST)
requires additional information on the correct way to address the VME board.
This information may be obtained from one of two sources. For a qualified VME
board, the VME board vendor may issue application notes on the correct use of
the board. Otherwise, refer to the *Guidelines for the Selection of Third-Party
VME Modules,* GFK-0448.

The VMEWRT function has four input parameters and one output parameter. When the
VMEWRT function receives power flow, the function copies the data from the input parameter IN
to the VME module at the address specified in ADR and the address modifier AM. The VMEWRT
function passes power to the right to indicate a successful transfer of data.

```
                              _____
                             |      |
        (enable)         ─| VME_ |─    (ok)
                             |      |
                             | WRT_ |
                             | WORD |
(input parameter IN) ─| IN   |
                             |  LEN |
                             |00001 |
   (address modifier) ─| AM   |
                             |      |
      (data address) ─| ADR  |
                             |_____|
```
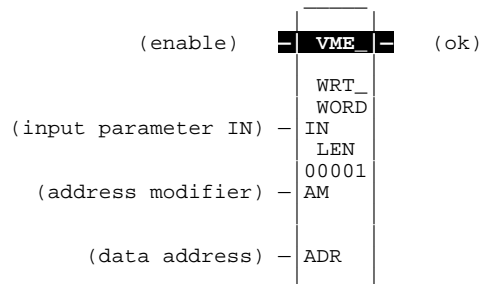
Parameters:

Parameter	Description
enable	When enable is energized, the VME write is performed.
IN	IN contains the data to be written to the address specified by ADR and AM.
AM	AM contains the address modifier.
ADR	ADR contains the address where the data is to be written.
LEN	LEN may be 1 to 32,767.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o		o		•	•	•	•	•	•	•	
AM	•	•	•	•	•		•		•	•	•	•	•	•	•	
ADR	•								•	•	•	•	•	•	•	
ok	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for BYTE data only; not valid for WORD.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, when enabling input %I00001 is ON, the hexadecimal value FFFF is written to each of 20 words on the VME bus, the first (lowest address) being specified by the content of %R00019 (low word) and %R00020 (high word). Unless an error occurs while writing the data, internal reference %M00055 is set ON.

```
  |           _____
  | %I00001  |     |                                                      %M00055
  |—| |——————| VME_|——————————————————————————————————————————————————————( )—
  |          | WRT_|
  |          | WORD|
  | CONST  — | IN  |
  | FFFF     | LEN |
  |          | 00020
  | CONST  — | AM  |
  | 00039    |     |
  |          |     |
  | %R00019— | ADR |
  |          |_____|
  |
```

VMERMW   (BYTE, WORD)

The VME Read/Modify/Write (VMERMW) function is used to update a data element on the VME bus.

Note

Using a VME function (VMERD, VMEWRT, VMERMW, or VMETST) requires additional information on the correct way to address the VME board. This information may be obtained from one of two sources. For a qualified VME board, the VME board vendor may issue application notes on the correct use of the board. Otherwise, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

The VMERMW function has five input parameters and one output parameter. When the VMERMW function receives power flow, the function reads the data from the board at the address specified by ADR and address modifier AM. This byte or word of data is combined (AND/OR) with the data mask MSK. Selection of AND or OR is made using the OP input. MSK is a word value. If byte data is operated on, only the lower 8 bits of MSK are used.

The result is then written back to the same VME address from which it was read. The VMERMW function passes power flow to the right whenever power is received unless an error occurs.

```
                               _____
                              |        |
         (enable)     ──────■|  VME_  |■──────    (ok)
                              |        |
                              | RMW_   |
                              | WORD   |
(VME operation)     ──────────|OP      |
                              |        |
    (data mask)     ──────────|MSK     |
                              |        |
(address modifier)  ──────────|AM      |
                              |        |
  (data address)    ──────────|ADR     |
                              |_____|
```
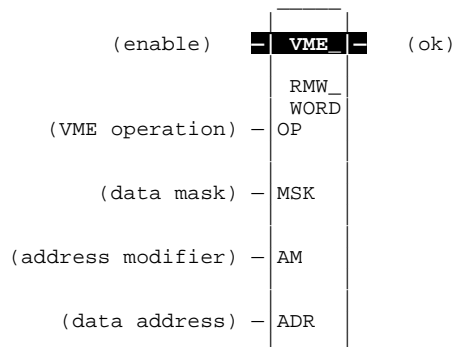
Parameters:

Parameter	Description
enable	When enable is energized, the VME function is performed.
OP	OP specifies whether data is to be ANDed or ORed with the MSK data.
MSK	MSK contains the data mask.
AM	AM contains the address modifier.
ok	The ok output is energized whenever the function is enabled and performed without error.

Valid Memory Types:

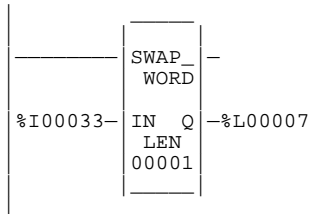| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
OP															•	
MSK	•	•	•	•	•		•		•	•	•	•	•	•	•	
AM	•	•	•	•	•		•		•	•	•	•	•	•	•	
ADR	•								•	•	•	•	•	•	•	
ok	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, when enabling input %M00044 is ON, the hexadecimal value 80H is ORed with the byte of data read from address 060010H on the VME bus in rack 0 (the main rack) using Standard Non-Privileged Data Access. Unless an error occurs while accessing the data, coil %Q00027 is set ON.

```
|           _____
|%M00044   |      |                                              %Q00027
|—| |——————| VME_ |——————————————————————————————————————————————( )—
|          | RMW_ |
|          | BYTE |
| CONST —  | OP   |
|  00001   |      |
|          |      |
| CONST —  | MSK  |
|  00080   |      |
|          |      |
| CONST —  | AM   |
|  00039   |      |
|          |      |
| CONST —  | ADR  |
| 00060010 |_____|
|
```

VMETST   (BYTE, WORD)

Use the VME Test and Set (VMETST) function to handle semaphores on the VME bus. The VMETST function exchanges a Boolean ON (1) for the value currently at the semaphore location. If that value was already ON, then the VMETST function does not obtain the semaphore. If the existing value was OFF, then the semaphore is reset and the VMETST function has the semaphore and the use of the memory area it controls. The semaphore is cleared using the VMEWRT function to write a 0 to the semaphore location.

Note

Using a VME function (VMERD, VMEWRT, VMERMW, or VMETST) requires additional information on the correct way to address the VME board. This information may be obtained from one of two sources. For a qualified VME board, the VME board vendor may issue application notes on the correct use of the board. Otherwise, refer to the *Guidelines for the Selection of Third-Party VME Modules*, GFK-0448.

The VMETST function has three input parameters and two output parameters. When the VMETST function receives power flow, a Boolean ON is exchanged with the data at the address specified by ADR using the address modifier specified by AM. The VMETST function sets the Q output to ON if the semaphore was available (OFF) and was acquired. The function passes power flow to the right whenever power is received and no error occurs during execution.

```
                         _____
                        |        |
      (enable)      ─── | VME_   | ───   (ok)
                        |        |
                        |  TS_   |
                        |  WORD  |
(address modifier) ─── | AM   Q | ───
                        |        |
                        |        |
        (address) ─── | ADR    |
                        |_____|
```
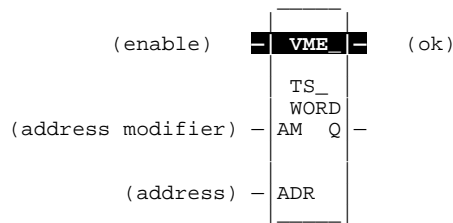
Parameters:

Parameter	Description
enable	When enable is energized, the VME test and set is performed.
AM	AM contains the address modifier.
ADR	ADR contains the address of the semaphore.
ok	The ok output is energized when the function is enabled and performed without error.
Q	Output Q is set ON if the semaphore was available (OFF). Otherwise, Q is set OFF.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
AM	•								•	•	•	•	•	•	•	
ADR	•								•	•	•	•	•	•	•	
ok	•															•
Q	•															•

<u>Note</u>:   Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the VMERD, VMEWRT, and VMETST functions are used to read data protected by a semaphore. When enabling input %M00047 is ON, the VMETST function is executed to obtain the semaphore. The semaphore VME address is stored in %R00041 and %R00042 using Standard Non-Privileged Data Access. When this is successful, coil %M00047 is reset and coil %M00048 is set. When %M00048 is set, the VMERD function reads the data (20 words of data whose VME address is stored in %R00043 and %R00044, data read into %R00200 through %R00219). When the read is successful (something is broken or not programmed correctly - if it is not), the VMTWRT function relinquishes the semaphore. Coil %M00048 is reset when the VMEWRT is successful. %M00049 is set to indicate that fresh data is now available.

If the semaphore is not available, VMERD and VMEWRT are not executed. The effect is that setting %M00047 causes the PLC to check the semaphore each sweep until the semaphore is available. When it becomes available, the semaphore is acquired, the data is read, and the semaphore is relinquished. No further action is taken until %M00047 is set again.

```
 |
 |%M00047     _____
 |—| |——— |  VME_  |—
 |           |  TS_  |
 |           |  BYTE |
 |           |       |                                                    (RM)—
 | CONST —— | AM   Q |
 |  0039     |       |            |                                    %M00048
 |           |       |            —————————————————————————————————————(SM)—
 |%R00041— | ADR   |
 |           |_____|
 |
 |
 |%M00048     _____                  _____                           %M00048 |——
 |—| |——— |  VME_  |———————————————— |  VME_  |——————————————————————(RM)—
 |           |  RD _ |                 |  WRT_ |          |
 |           |  WORD |                 |  BYTE |                        %M00049 | |
 | CONST —— | AM    |        CONST —— | IN    |          —————————————————————(SM)—
 |  0039     |  LEN  |         0000    |  LEN  |
 |           | 00020 |                 | 00001 |
 |%R00043— | ADR  Q |—%R00200  CONST —— | AM    |
 |           |_____|          0039    |       |
 |                                      |       |
 |                           %R00041— | ADR   |
 |                                      |_____|
 |
```
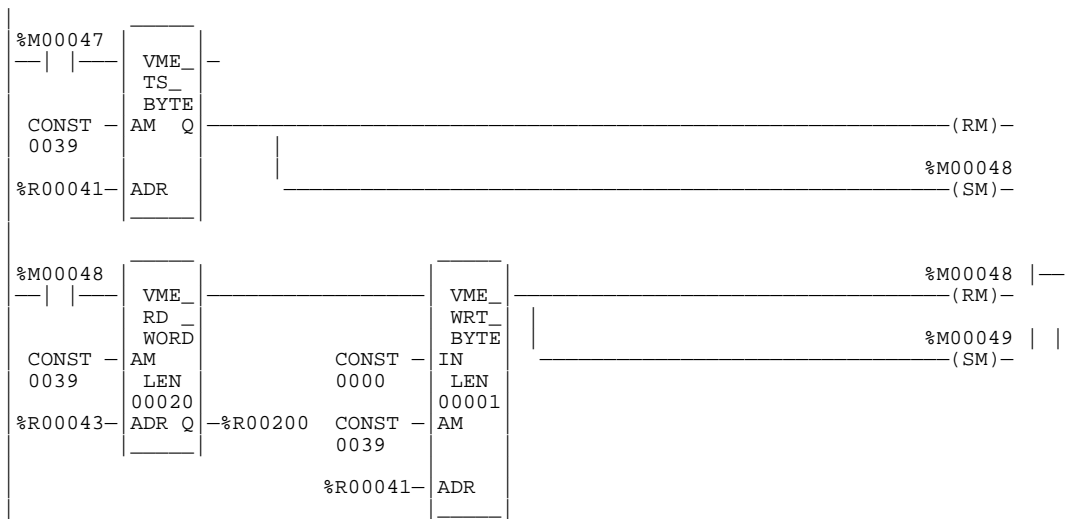
VME_CFG_RD

Use the VME_CFG_RD function to read data from the VME bus. The VME_CFG_RD function has five input parameters and three output parameters. When the function receives power, the data elements (N) are read from the VME bus at the location defined by rack (R), slot (S), and dual port offset (OFF). The data read is placed in output Q. The status of the operation is placed in the status word output (ST). The function has a length specification (LEN) of the maximum size of the output array.

Note

The module at the specified rack and slot must be configured as a third-party VME module in BUS INTERFACE mode for this function block to execute successfully. Additional parameters indicating the AM code, the location and size of the module's dual port, and the bus interface type must be specified in the module's configuration for correct operation. See chapter 11 of the *Logicmaster 90-70 Programming Software User's Manual* (GFK-0263) for more information on configuration of third-party VME modules.

If the function is completed successfully, ok is set ON; otherwise, it is set OFF. It is also set OFF when:

- The number of data elements (N) is greater than the length (LEN) specified.

- The rack/slot value (R and S) is out of range or is not a valid VME location.

- The most significant byte of the dual port offset (OFF) is not zero.

- The most significant byte of the dual port address plus the dual port offset is not zero.

- Read beyond the end of dual port memory.

- Specified rack/slot not configured for a Third-Party VME module in **BUS INTERFACE** mode.

- If the dual port offset is an even number, configure for the odd byte only. If the dual port offset is an odd number, configure for word or single word.

```
                            _____
         (enable)  —| VME |—    (ok)
                     |        |
                     | CFG_  |
                     | READ  |
    (rack number) —| R   ST |— (status word)
                     | LEN   |
                     | 00001 |
    (slot number) —| S    Q |— (output parameter Q)
                     |        |
(dual port offset) —| OFF    |
                     |        |
  (data elements) —| N      |
                     |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the data initialization is performed.
R	The rack number is specified in R.
S	The slot number is specified in S.
OFF	OFF specifies the dual port offset.
N	N contains the amount of data (data elements) to be read from the VME bus.
ok	The ok output is energized when the function is performed without error.
ST	The status word contains the status of the operation.
Q	When the function is performed, the data is read to array Q.
LEN	LEN is the length of the output array in bytes.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
R	•	•	•	•	•	•	•		•	•	•	•	•	•	•	
S	•	•	•	•	•	•	•		•	•	•	•	•	•	•	
OFF	•					•			•	•	•	•	•	•	•	
N	•	•	•	•	•	•	•		•	•	•	•	•	•	•	
ok	•															•
ST	•	•	•	•	•	†	•		•	•	•	•	•	•		
Q	•	•	•	•	•	†	•		•	•	•	•	•	•		

•   Valid reference or place where power may flow through the function

†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
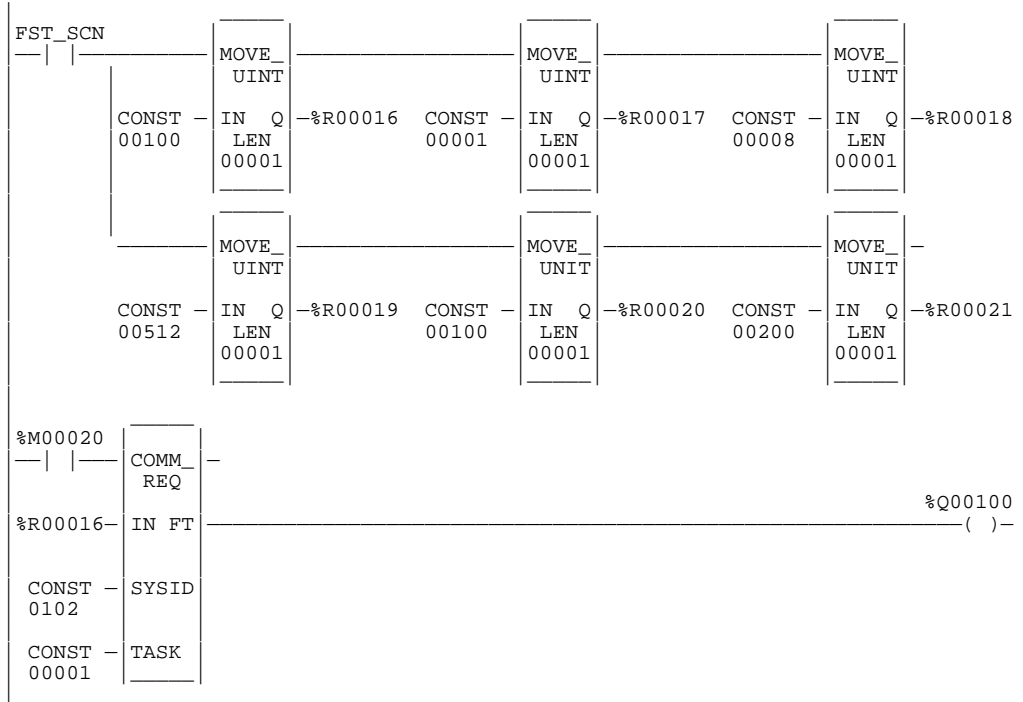
Example:

In the following example, when enable is ON, VME data at rack 1, slot 3 and dual port offset defined by %R00100 is read into %R00101 through %R00110 of the array %R00101 through %R00116. If an error was encountered, the status word %AQ0001 will contain an error code.

```
 |          _____
 |%I00001  |     |
 |—| |————| VME_ |—
 |         | CFG_ |
 |         | READ |
 | CONST —|R  ST |—%AQ0001
 | 00001  | LEN  |
 |        | 00016|
 | CONST —|S   Q |—%R00101
 | 00003  |      |
 |        |      |
 |%R00100—| OFF  |
 |        |      |
 |        |      |
 | CONST —|N     |
 | 00010  |_____|
 |
```

VME_CFG_WRITE

Use the VME_CONFIG_WRITE function to write data from the VME bus. The VME_CFG_WRT function has six input parameters and two output parameters. When the function receives power, the data elements (N) are written from the data array (IN) to the VME bus at the location defined by rack (R), slot (S), and dual port offset (OFF). The status of the operation is placed in the status word output (ST). The function has a length specification (LEN) of the maximum size of the output array.

Note

The module at the specified rack and slot must be configured as a third-party VME module in BUS INTERFACE mode for this function block to execute successfully. Additional parameters indicating the AM code, the location and size of the module's dual port, and the bus interface type must be specified in the module's configuration for correct operation. See chapter 11 of the *Logicmaster 90-70 Programming Software User's Manual* (GFK-0263) for more information on configuration of third-party VME modules.

If the function is completed successfully, ok is set ON; otherwise, it is set OFF. It is also set OFF when:

- The number of data elements (N) is greater than the length (LEN) specified.

- The rack/slot value (R and S) is out of range or is not a valid VME location.

- The most significant byte of the dual port offset (OFF) is not zero.

- The most significant byte of the dual port address plus the dual port offset is not zero.

- Read beyond the end of dual port memory.

- Specified rack/slot not configured for a Third-Party VME module in **BUS INTERFACE** mode.

- If the dual port offset is an even number, configure for the odd byte only. If the dual port offset is an odd number, configure for word or single word.

```
                                        _____
            (enable)  ─  █  VME_  █       (ok)

                            CFG_
                           WRITE
(input parameter IN)  ─  IN  ST  ─  (status word)
                           LEN
                          00001
       (rack number)  ─  R

       (slot number)  ─  S

  (dual port offset)  ─  OFF

    (data elements)  ─  N
                         _____
```

Parameters:

Parameter	Description
enable	When the function is enabled, the data initialization is performed.
IN	IN contains the data to be written to the VME bus at the location defined by rack (R), slot (S), and dual port offset (OFF).
R	The rack number is specified in R.
S	The slot number is specified in S.
OFF	OFF specifies the dual port offset.
N	N contains the amount of data (data elements) to be written to the VME bus.
ok	The ok output is energized when the function is performed without error.
ST	The status word contains the status of the operation.
LEN	LEN is the length of the input array in bytes.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	•	•	•	•	•	•		•	•	•	•	•	•		
R	•	•	•	•	•	•	•		•	•	•	•	•	•	•	
S	•	•	•	•	•	•	•		•	•	•	•	•	•	•	
OFF	•					•			•	•	•	•	•	•	•	
N	•	•	•	•	•	•	•		•	•	•	•	•	•	•	
ok	•															•
ST	•	•	•	•	•	†	•		•	•	•	•	•	•		

•   Valid reference or place where power may flow through the function
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, when enable is ON, data from %R00101 through %R00110 of the array %R00101 through %R00116 is written to the VME bus at rack 1, slot 3 and dual port offset defined by %R00100. If an error was encountered, the status word %AQ0001 will contain an error code.

```
 |              _____
 | %I00001    |     |
 |—| |———|    | VME_|—
 |            | CFG_|
 |            |WRITE|
 | %R00101—|IN ST|—%AQ00001
 |            | LEN |
 |            |00016|
 | CONST —|R   |
 | 00001    |     |
 |            |     |
 | CONST —|S   |
 | 00003    |     |
 |            |     |
 | %R00100—|OFF |
 |            |     |
 | CONST —|N   |
 | 00010    |_____|
 |
```

DATA_INIT   (INT, UINT, DINT, WORD, DWORD, REAL)

Use the Data Initialization (DATA_INIT) function to copy a block of constant data to a reference range.

The DATA_INIT function has one input parameter and two output parameters. When the function receives power flow, it copies the constant data to output Q. The function's constant data length (LEN) specifies how much constant data of the function type is copied to consecutive reference addresses starting at output Q.

Note

The output parameter is not included in coil checking.

The function passes power to the right whenever power is received.

```
                    _____
(enable)         ─█─|DATA_ |─█─   (ok)
                    |        |
                    |INT_    |
                    |WORD    |
                    |     Q  |─  (output parameter Q)
                    |LEN     |
                    |        |
                    |00001   |
                    |_____|
```

Note

When the DATA_INIT instruction is first programmed, the constant data is initialized to zeroes. The constant data may be changed by zooming into the function (see below for details).

Parameters:

Parameter	Description
enable	When the function is enabled, the data initialization is performed.
ok	The ok output is energized whenever the function is enabled.
Q	When the data initialization is performed, the constant data is written to Q.
LEN	LEN specifies how much constant data is copied to consecutive reference addresses starting at output Q.

Valid Memory Types:

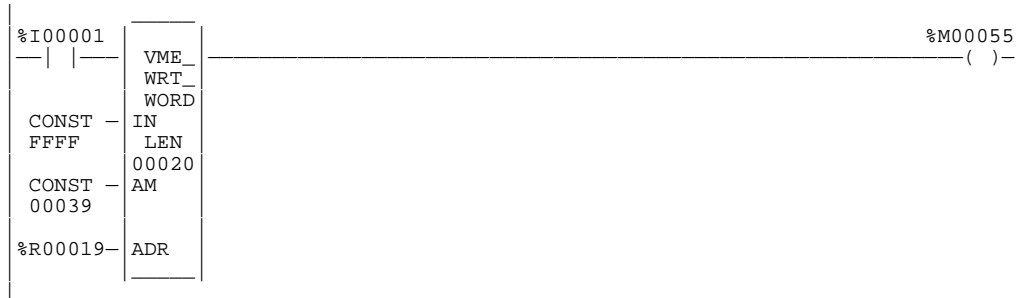| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
ok	•															•
Q		o	o	o	o		o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, or WORD data only..

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, on the first scan (FST_SCN) 100 words of initial data is copied to %R00005 through %R00104.

```
         |        _____
         | FST_SCN|      |
         |—| |————| DATA_|—
         |        | INIT—|
         |        |  INT |
         |        |     Q|—%R00005
         |        | LEN  |
         |        | 00100|
         |        |_____|
         |
```

Zooming into the DATA_INIT_type Function Block

In Logicmaster, press **Zoom (F10)** to zoom into the DATA_INIT_type function block.

The DATA_INIT_type window contains a *New Value* field which functions as a mini command line, an *Element* field which indicates which data element the cursor is currently positioned on, a *Length* field which tells how many elements the DATA_INIT_type instruction contains, and the block of constant data. The *Element* and *Length* fields cannot be edited.

To change a value, position the cursor on the element to be changed, and enter the new value in the *New Value* field.

To help you position the cursor, the window contains starting element line labels to the left of the data block. The *Element* field will contain the element number the cursor is currently positioned on.

The data block portion of the window will scroll and page down.

DATA_INIT_COMM

Use the Data Initialize Communications Request (DATA_INIT_COMM) function to initialize a COMMREQ function with a block of constant data. The IN parameter of the COMMREQ must correspond with output Q of this DATA_INIT_COMM function.

The DATA_INIT_COMM function has one input parameter and two output parameters. When the function receives power flow, it copies the constant data to output Q. The function's constant data length (LEN) specifies how many words of constant data are to be copied to consecutive reference addresses starting at output Q. The length should be equal to the size of the COMMREQ function's entire command block.

Note

The output parameter is not included in coil checking.

The function passes power to the right whenever power is received.

```
                        _____
                       |      |
(enable)          ─ | DATA_ | ─      (ok)
                       | INIT_ |
                       | COMM |
                       |    Q | ─ (output parameter Q)
                       | LEN  |
                       |00001 |
                       |_____|
```

Note

When the DATA_INIT instruction is first programmed, the constant data is initialized to zeroes. The constant data may be changed by zooming into the function (see below for details).

Parameters:

Parameter	Description
enable	When the function is enabled, the data initialization is performed.
ok	The ok output is energized whenever the function is enabled.
Q	When the data initialization is performed, the constant data is written to Q.
LEN	LEN specifies how many words of constant data are to be copied to consecutive reference addresses starting at output Q. LEN must equal the size of the COMMREQ function's entire command block.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
ok	•															•
Q									•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, on the first scan (FST_SCN) a command block consisting of 100 words of data and 6 words of header is copied to %P00001 through %P00099; and %Q00002 will receive power.

```
         _____
|FST_SCN |      |                                                    %Q00002
—| |——|DATA_ |————————————————————————————————————————————————————————( )—
|        |INIT— |
|        | COMM |
|        |    Q |—%P00001
|        |LEN   |
|        |00100 |
|        |_____|
|
```

Zooming into the DATA_INIT_COMM Function Block

In Logicmaster, press **Zoom (F10)** to zoom into the DATA_INIT_COMM function block.

The DATA_INIT_COMM window contains a *New Value* field for entering values and a *Data Element* field, which indicates which data element in the data block the cursor is currently positioned on. Only the *Data Element* field cannot be changed.

The window also contains the first six words of the COMMREQ command block, with labels consistent with each word's use and the COMMREQ data block. The first six words are *Data Length, Wait flag, Timeout, Status Memory, Status Offset, and Max comm time*.

The line labels to the left of the data block elements and the *Data Element* field will correspond to the data block portion of the COMMREQ block. For example, the data block elements will be labeled beginning with number 1 even though the data block begins at the seventh word of the instruction.

The data block portion of the window will scroll and page down.

Note

The data block information is in hexadecimal.

DATA_INIT_ASCII

Use the Data Initialize ASCII (DATA_INIT_ASCII) function to copy a block of constant ASCII text to a reference range.

The DATA_INIT_ASCII function has one input parameter and two output parameters. When the function receives power flow, it copies the constant data to output Q. The function'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.

Note

The output parameter is not included in coil checking.

The function passes power to the right whenever power is received.

```
                         _____
(enable)           ─ ■ DATA_ ■ ─    (ok)
                      │       │
                      │ INIT_ │
                      │ ASCII │
                      │     Q │─ (output parameter Q)
                      │  LEN  │
                      │ 00001 │
                      │_____│
```
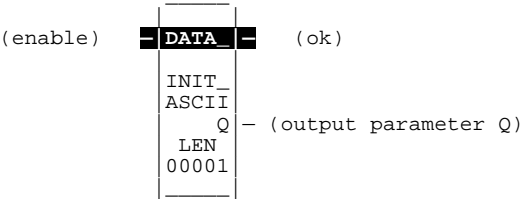
Note

When the DATA_INIT instruction is first programmed, the constant data is initialized to blanks. The constant data may be changed by zooming into the function (see below for details).

Parameters:

Parameter	Description
enable	When the function is enabled, the data initialization is performed.
ok	The ok output is energized whenever the function is enabled.
Q	When the data initialization is performed, the constant data is written to Q.
LEN	LEN specifies how many bytes of constant text are copied to consecutive reference addresses starting at output Q. LEN must be an even number.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
ok	•															•
Q		•	•	•	•		•		•	•	•	•	•	•		

Note   Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, on the first scan (FST_SCN) the decimal equivalent of 100 bytes of ASCII text is copied to %R00050 through %R00149; and %Q00002 will receive power.

```
                    _____
 |FST_SCN         |      |                                                    %Q00002
 |—| |————| DATA_ |————————————————————————————————————————————————————————————( )—
 |              | INIT_ |
 |              | ASCII |
 |            Q |—%R00050
 |              | LEN   |
 |              | 00100 |
 |              |_____|
 |
```

Zooming into the DATA_INIT_ASCII Function Block

In Logicmaster, press **Zoom (F10)** to zoom into the DATA_INIT_ASCII function block.

The DATA_INIT_ASCII window contains a mini command line, a *Function Len* field which indicates how many character bytes the instruction contains, an *Element No.* field which indicates which character byte the cursor is currently positioned on, and the block of ASCII text.

ASCII text is entered by positioning the cursor on the starting byte and typing a quotation mark (") followed by the desired string on the mini command line (e.g., "This is a test). (Refer to the example on the next page.)

You can enter as many characters as will fit on the command line at one time. Enter non-printable characters with a backslash immediately followed by the 3-digit decimal equivalent of the character (e.g., "\010 for line feed).

After entering the text on the command line, press the **Enter** key.

Press the **Escape** key to exit this window and return to the ladder diagram logic display.

DATA_INIT_DLAN

The DATA Initialize DLAN (DATA_INIT_DLAN) function is for use with a DLAN system which is a limited availability, specialty system. If you have a DLAN system, refer to the *Series 90™-70 DLAN/DLAN+ Interface Module User's Manual* (GFK-0729B) for details.

Note

This module is not available as a general purchase item.

Data Table Functions

Data tables provide automatic data move capabilities. The data table functions are used to enter values into or copy values out of a table. To use these functions properly, the program must include other logic to move data through the tables and to initialize table pointers.

Moving Values In and Out of a Table

All of the table write functions fill a table with values that are read from a specified reference location. Similarly, all of the table read functions copy values from a table to a specified reference location.
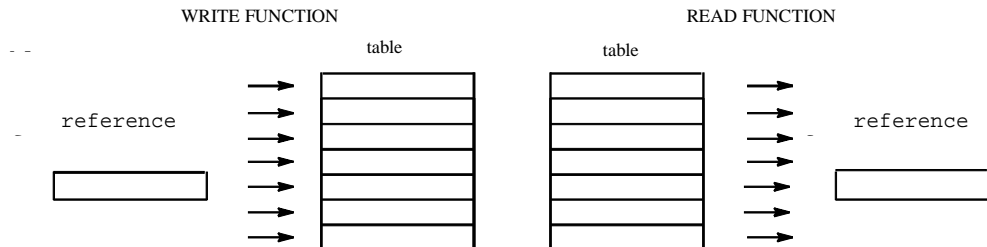
One value is written or read each time these functions are called. Other logic in the program must be used to place new values in the reference for a write function, or to capture values from the reference after a read function.

Initializing the Table Pointer

The read and write functions keep track of the current table location by means of a pointer. Additional program logic must be used to initialize this value so that the function will begin reading or writing the table at the correct location. Normally, the value in this reference is initialized to zero.

This chapter describes the following data table functions.

Abbreviation	Function	Description	Page
TBLRD	Table Read	Copy a value from a specified table location to an output reference.	10-3
TBLWR	Table Write	Copy a value from an input reference to a specified table location.	10-5
LIFORD	LIFO Read	Remove the entry at the pointer location, and decrement the pointer by one. LIFORD is used in conjunction with LIFOWRT (see below).	10-7
LIFOWRT	LIFO Write	Increment the table pointer and write data to the table. LIFOWRT is used in conjunction with LIFORD (see above).	10-9
FIFORD	FIFO Read	Remove the entry at the bottom of the table, and decrement the pointer by one. FIFORD is used in conjunction with FIFOWRT (see below).	10-11
FIFOWRT	FIFO Write	Increment the table pointer and write data to the table. FIFOWRT is used in conjunction with FIFORD (see above).	10-13
SORT	Sort	Sort an array in ascending order.	10-15
ARRAY_MOVE	Array Move	Copy a specified number of data elements from a source array to a destination array.	10-17
SRCH_EQ	Search Equal	Search for all array values equal to a specified value.	10-21
SRCH_NE	Search Not Equal	Search for all array values not equal to a specified value.	10-21
SRCH_GT	Search Greater Than	Search for all array values greater than a specified value.	10-21
SRCH_GE	Search Greater Than or Equal	Search for all array values greater than or equal to a specified value.	10-21
SRCH_LT	Search Less Than	Search for all array values less than a specified value.	10-21
SRCH_LE	Search Less Than or Equal	Search for all array values less than or equal to a specified value.	10-21
ARRAY_RANGE	Array Range	Determine if a value is between the range specified in two tables.	10-24

All values in the table must be the same type, which may be signed integer (INT), double precision signed integer (DINT), unsigned integer (UINT), word (WORD), or double word (DWORD). The BYTE data type is available for SRCH_EQ, SRCH_NE, SRCH_GT, SRCH_LT, SRCH_GE, SRCH_LE, and ARRAY_MOVE; and BIT is also available for ARRAY_MOVE.

Note

No REAL data type exists for the data table functions; however, the DWORD data type may be used to manipulate real data.

The maximum length allowed for each data table function is 32,767 elements, except for the SORT function which is restricted to 64 elements.

TBLRD   (INT, UINT, DINT, WORD, DWORD)

The Table Read (TBLRD) function is used to sequentially read values in a table which never becomes full. When the pointer reaches the end of the table, it automatically wraps around to the beginning of the table.
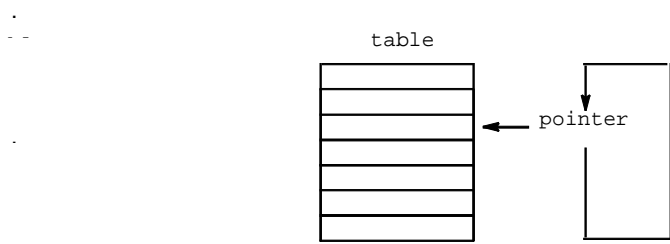
1. The function increments the pointer by one.

2. The function 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 table is empty (PTR = LEN).

4. When the table is full, the pointer wraps around to the beginning of the table.

Note

> TBLRD and TBLWRT 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.

The TBLRD function has three input parameters and three output parameters. When the function 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 the function executes, the pointer will automatically be set back to 1. After the pointer is incremented, the content at the new pointer location is copied to output Q.

The function always passes power to the right when power is received.

```
                         _____
      (enable)     ─|  TBL_ |─    (ok)
                     |       |
                     | RD_   |
                     | WORD  |
(table elements) ─| TB EM |─ (output parameter EM)
                     | LEN   |
                     | 00001 |
       (pointer) ─| PTR Q |─ (output parameter Q)
                     |_____|
```
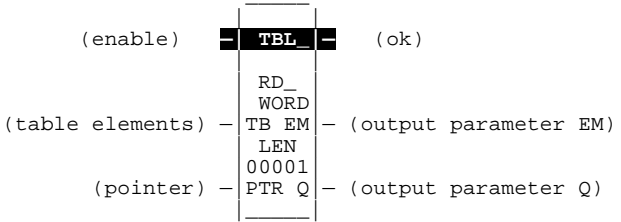
Parameters:

Parameter	Description
enable	When enable is energized, the table read is performed.
TB	TB contains the elements in the table.
PTR	PTR is incremented; it then points to the next table element to be read.
ok	The ok output is energized whenever the function is enabled.
EM	Output EM is energized when the last element of the table is read.
Q	Output Q contains the element read from the table.
LEN	LEN must be between 1 and 32,767.

Valid Memory Types:

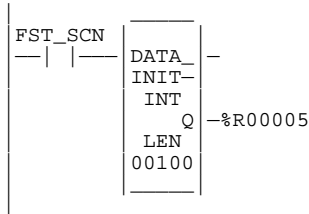| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
TB	•	o	o	o	o	Δ	o		•	•	•	•	•	•		
PTR		•	•	•	•		•		•	•	•	•	•	•		
ok	•															•
EM	•															•
Q	•	o	o	o	o	Δo	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.
o Valid reference for INT, UINT, or WORD data only; not valid for DINT or DWORD.
Δ Valid reference for WORD data only; not valid for INT, UINT, DINT, or DWORD.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following 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 is 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.

```
          _____
 %M00346 |      |
—| |—————| TBL_ |—
         |  RD_ |
         |  INT |                                                    %M01001
WIDGETS—|TB  EM|————————————————————————————————————————————————————( )—
         | LEN  |
         |00020 |
 %L00001—|PTR  Q|—ITEM_CT
         |_____|
```
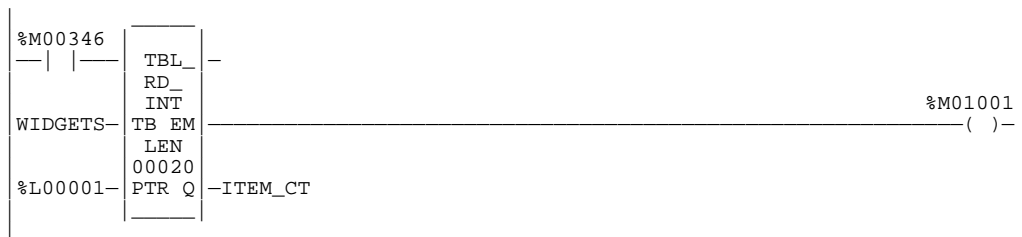
TBLWRT   (INT, UINT, DINT, WORD, DWORD)

The Table Write (TBLWRT) function is used to sequentially update values in a table that never becomes full. When the pointer reaches the end of the table, it automatically returns to the beginning of the table.
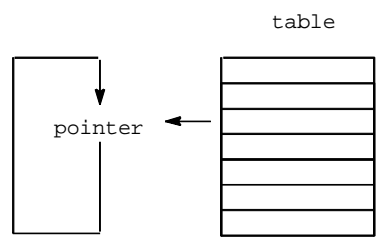
table

pointer

1. **T**he function increments the pointer by one.

2. The function copies data from input parameter IN to the position in the table indicated by the pointer. (It will write 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).

4. When the table is full, the pointer wraps around to the beginning of the table.

Note

TBLWRT and TBLRD 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.

The TBLWRT function has four input parameters and two output parameters. When the function 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 the function executes, the pointer will automatically be set back to 1. After incrementing the pointer, the TBLWRT function writes the content of the input reference to the current pointer location, overwriting data already stored there.

The function always passes power to the right when power is received.

```
                          ┌─────┐
        (enable) ──┤ TBL_ ├──   (ok)
                      │     │
                      │WRT_ │
                      │WORD │
(table elements) ──│TB FL├── (output parameter FL)
                      │ LEN │
                      │00001│
        (pointer) ──│PTR  │
                      │     │
                      │     │
(input parameter IN) ──│IN   │
                          └─────┘
```
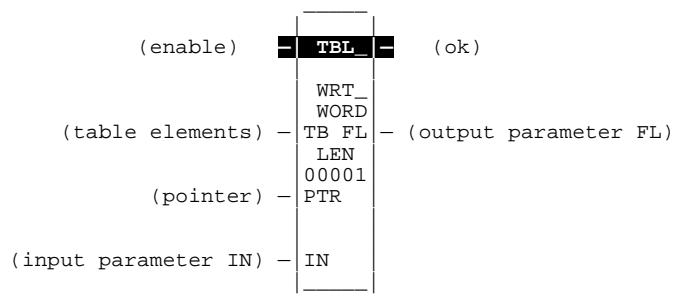
Parameters:

Parameter	Description
enable	When enable is energized, the table write is performed.
TB	TB contains the elements in the table.
PTR	PTR is incremented; it then points to the next table element to be written.
IN	IN contains the element to be written to the table.
ok	The ok output is energized whenever the function is enabled.
FL	Output FL is energized when IN is written to the last element of the table.
LEN	LEN must be between 1 and 32,767.

Valid Memory Types:

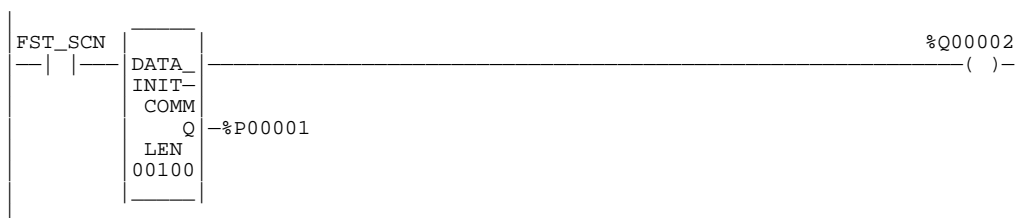| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
TB		o	o	o	o	Δ†	o		•	•	•	•	•	•		
PTR		•	•	•	•		•		•	•	•	•	•	•		
IN	•	o	o	o	o	Δ	o		•	•	•	•	•	•	•	
ok	•															•
FL	•															•

<u>Note</u>:   Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, or WORD data only; not valid for DINT or DWORD.
Δ   Valid reference for WORD data only; not valid for INT, UINT, DINT, or DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, WIDGETS is a table with 20 integer elements. When the enabling input %I00012 is ON, the pointer increments and the contents of %P00077 are written into the table at the pointer location. %L00001 functions as the pointer into the data table.

```
           ┌──────┐
%I00012    │      │─
──| |──────│ TBL_ │
           │ WRT_ │
           │ INT  │                                              %M01001
WIDGETS────│TB  FL│──────────────────────────────────────────────( )─
           │ LEN  │
           │00020 │
%L00001────│PTR   │
           │      │
           │      │
%P00077────│IN    │
           │      │
           └──────┘
```

LIFORD   (INT, UINT, DINT, WORD, DWORD)

The Last-In-First-Out (LIFO) Read (LIFORD) function is used to move 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, the LIFORD function must be used to remove the entry at the pointer location and decrement the pointer by one. The LIFORD function is used in conjunction with the LIFOWRT function, which increments the pointer and writes entries into the table.
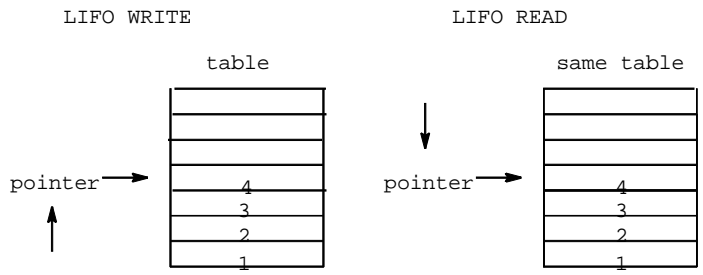
```
        LIFO WRITE                       LIFO READ

            table                        same table
          ┌──────────┐                  ┌──────────┐
          ├──────────┤                  ├──────────┤
          ├──────────┤        │         ├──────────┤
          ├──────────┤        │         ├──────────┤
pointer ─→├────4─────┤        ▼  pointer─→├────4─────┤
          ├────3─────┤                  ├────3─────┤
          ├────2─────┤                  ├────2─────┤
          ├────1─────┤                  ├────1─────┤
      ↑   └──────────┘                  └──────────┘
      │
```

Sequence of events:

1. The function 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. The function 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).

4. The pointer does not wrap around when the table is full.

The LIFORD function has three input parameters and three output parameters. When the function 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. Therefore, EM indicates whether or not the table is empty. If the table is empty when the function receives power flow, no read will occur. The pointer always indicates the last item entered into the table.

The function passes power to the right if the pointer was in range for an element to be read.

```
                           ┌─────┐
        (enable) ──────────┤ LIFO_├──────── (ok)
                           │      │
                           │ RD_  │
                           │ WORD │
(table elements) ──────────┤TB  EM├──────── (output parameter EM)
                           │ LEN  │
                           │00001 │
        (pointer) ─────────┤PTR  Q├──────── (output parameter Q)
                           └─────┘
```
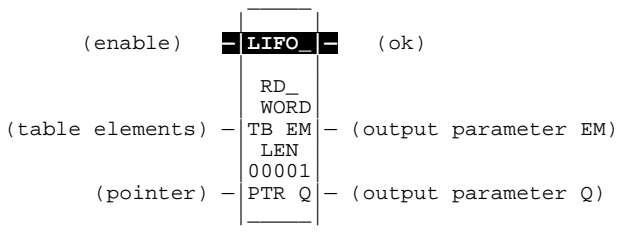
Parameters:

Parameter	Description
enable	When enable is energized, the read operation is performed.
TB	TB contains the elements in the table.
PTR	PTR points to the next LIFO element to be read; after the read, it is decremented.
ok	The ok output is energized when EN is ON and 0 < PTR ≤ LEN.
EM	Output EM is energized when the last element is read.
Q	Output Q contains the element read from the table.
LEN	LEN must be between 1 and 32,767.

Valid Memory Types:

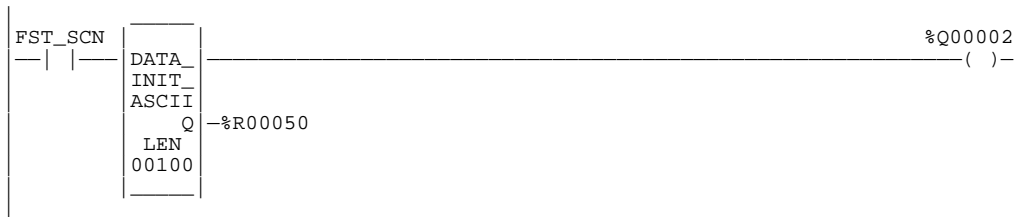| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
TB	•	o	o	o	o	o	Δ†	o		•	•	•	•	•	•	
PTR		•	•	•	•		•		•	•	•	•	•			
ok	•															•
EM	•															•
Q	•	o	o	o	o	Δ†	o		•	•	•	•	•	•		•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, or WORD data only; not valid for DINT or DWORD.
Δ   Valid reference for WORD data only; not valid for INT, UINT, DINT, or DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, 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.

```
         | ‾‾‾‾‾ |
PACK_IT  |       |
——| |——— | LIFO_ |—
         |  RD_  |
         | WORD  |                                                          EMPTY
PRODUCT— | TB EM |——————————————————————————————————————————————————————————( )—
         |  LEN  |
         | 00100 |
STK_PTR— | PTR Q |—CART
         | ‾‾‾‾‾ |
```

LIFOWRT (INT, UINT, DINT, WORD, DWORD)

The Last-In-First-Out (LIFO) Write (LIFOWRT) function is used to increment the table pointer by one, and then add an entry above the 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, no further values can be added by the LIFOWRT function. The LIFORD function must then be used to remove the entry at the pointer location and decrement the pointer by one.

```
          LIFO WRITE                        LIFO READ

              table                        same table
            ┌─────────┐                   ┌─────────┐
            ├─────────┤                   ├─────────┤
            ├─────────┤        │          ├─────────┤
pointer ───▶├────4────┤        │ pointer ─▶├────4────┤
            ├────3────┤        ▼          ├────3────┤
     ▲      ├────2────┤                   ├────2────┤
     │      ├────1────┤                   ├────1────┤
            └─────────┘                   └─────────┘
```
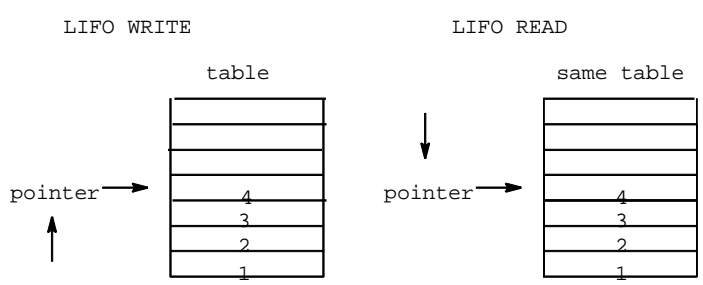
Sequence of events:

1. The function increments the table pointer by one.

2. The function copies data from input parameter IN to the position in the table indicated by the pointer. (It will write 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).

4. The pointer does not wrap around when the table is full.

The LIFOWRT function has four input parameters and two output parameters. When the function 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 the function 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 be possible to add more entries to it.

The function passes power to the right after a successful execution.
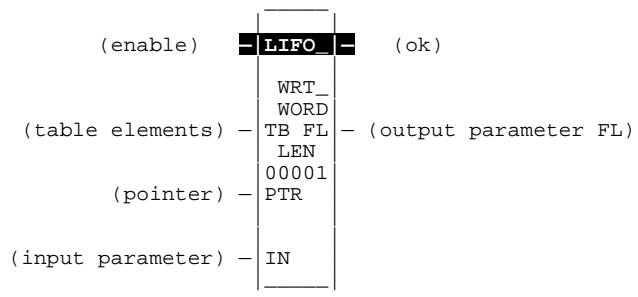
```
                            ┌─────────┐
       (enable)         ─── │ LIFO_   │ ─── (ok)
                            │  WRT_   │
                            │  WORD   │
(table elements) ─── TB FL  │ ─── (output parameter FL)
                            │  LEN    │
                            │ 00001   │
       (pointer) ─── PTR    │         │
                            │         │
(input parameter) ─── IN    │         │
                            └─────────┘
```

Parameters:

Parameter	Description
enable	When enable is energized, the write operation is performed.
TB	TB contains the elements in the table.
PTR	PTR is incremented; it then points to the next element to be written.
IN	IN contains the element to be written to the table.
ok	The ok output is energized when the function is enabled and PTR < LEN.
FL	Output FL is energized when the last element has been written.
LEN	LEN must be between 1 and 32,767.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
TB		o	o	o	o	Δ†	o		•	•	•	•	•	•		
PTR		•	•	•	•		•		•	•	•	•	•	•		
IN	•	o	o	o	o	Δ	o		•	•	•	•	•	•	•	
ok	•															•
FL	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
- • Valid reference or place where power may flow through the function.
- o Valid reference for INT, UINT, or WORD data only; not valid for DINT or DWORD.
- Δ Valid reference for WORD data only; not valid for INT, UINT, DINT, or DWORD.
- † %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, 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 node FL will pass 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 LIFORD function.

```
         |        _____
  STORE  |       |
——| |————| LIFO_ |—
         |  WRT_  |
         |  WORD  |                                              FULL
PRODUCT—| TB   FL |————————————————————————————————————————————( )—
         |  LEN   |
         | 00100  |
STK_PTR—| PTR     |
         |        |
         |        |
NEW_ITM—| IN      |
         |_____|
```
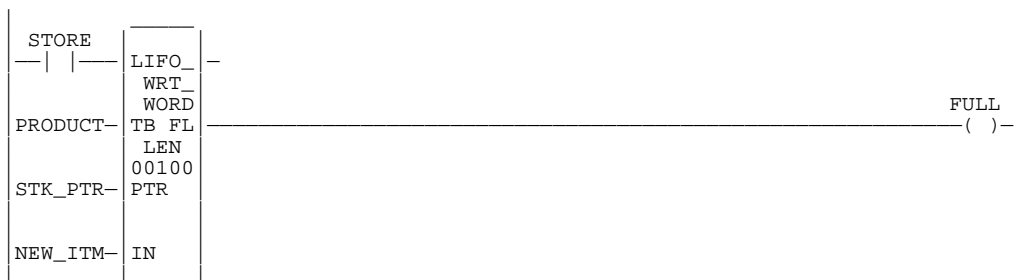
FIFORD   (INT, UINT, DINT, WORD, DWORD)

The First-In-First-Out (FIFO) Read (FIFORD) function is used to move 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, the FIFORD function must be used to remove the entry at the pointer location and decrement the pointer by one. The FIFORD function is used in conjunction with the FIFOWRT function, which increments the pointer and writes entries into the table.

```
        FIFO WRITE                        FIFO READ

            table                         same table

                                 |
                                 |
                                 |
                                 v
pointer --->        5      pointer --->        5
                    4                           4
        ^           3                           3
        |           2                           2
        |           1
```
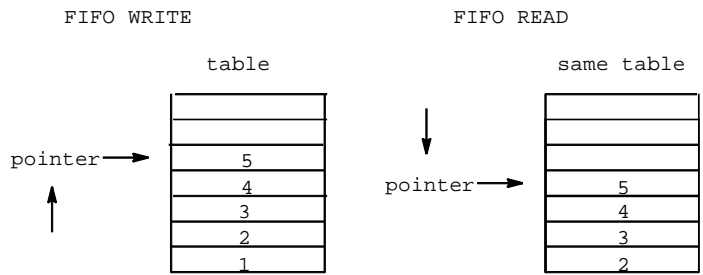
Sequence of events:

1. The function copies the top location 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. The function decrements the pointer by one.

4. Steps 1 and 2 are repeated each time the instruction is executed, until the table is empty (PTR = LEN).

5. The pointer does not wrap around when the table is full.

The FIFORD function has three input parameters and three output parameters. When the function 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. Therefore, EM indicates whether or not the table is empty.

The FIFORD function passes power to the right if the pointer is greater than zero and less than the value specified for LEN.

```
                         _____
       (enable)        _| FIFO_ |_     (ok)
                        |        |
                        | RD_    |
                        | WORD   |
(table elements) ------ | TB  EM | ---- (output parameter EM)
                        | LEN    |
                        | 00001  |
       (pointer) ------ | PTR  Q | ---- (output parameter Q)
                        |_____|
```
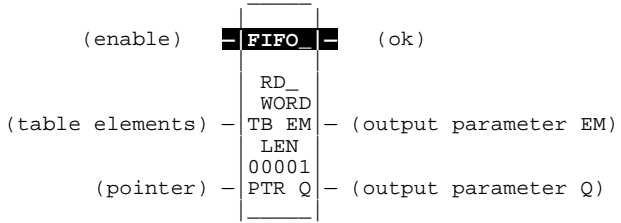
Parameters:

Parameter	Description
enable	When enable is energized, the FIFO read is performed. It is always the first element of the FIFO table that is read.
TB	TB contains the elements of the FIFO table.
PTR	PTR points to the last element of the FIFO table.
ok	The ok output is energized when the function is enabled and $0 < PTR < LEN$.
EM	Output EM is energized when the final element is read.
Q	Output Q contains the element read from the FIFO table.
LEN	LEN must be between 1 and 32,767.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
TB	•	o	o	o	o	Δ	o		•	•	•	•	•	•		
PTR		•	•	•	•		•		•	•	•	•	•	•		
ok	•															•
EM	•															•
Q	•	o	o	o	o	Δ†	o		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, or WORD data only; not valid for DINT or DWORD.
Δ   Valid reference for WORD data only; not valid for INT, UINT, DINT, or DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, refer to the "Restrictions on Formal Parameters within a Parameterized Subroutine Block" section of Chapter 2.

Example:

In the following example, 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 will be decremented. A copy of the oldest data item in the PRODUCT table will be left behind in each table location as the current data is copied out during successive PACK_IT triggers. Output node EM will pass 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 FIFOWRT function.

```
         |    _____
PACK_IT  |   |      |
—| |—————| FIFO_|—
         |   | RD_  |
         |   | WORD |                                              EMPTY
PRODUCT—|TB  EM|——————————————————————————————————————————————————( )—
         |   | LEN  |
         |   |00100 |
STK_PTR—|PTR Q|—CART
         |   |_____|
```

FIFOWRT (INT, UINT, DINT, WORD, DWORD)

The First-In-First-Out (FIFO) Write (FIFOWRT) function is used to increment the table pointer by one, and then add an entry at the new pointer location in a table. Values are always moved in at the bottom of the table. If the pointer reaches the last location and the table becomes full, no further values can be added by the FIFOWRT function. The FIFORD function must then be used to remove the entry at the pointer location and decrement the pointer by one.

```
         FIFO WRITE                    FIFO READ

              table                      same table

                                            |
 pointer ──→│     5     │                   ↓
            │     4     │    pointer ──→│     5     │
     ↑      │     3     │               │     4     │
     │      │     2     │               │     3     │
            │     1     │               │     2     │
```

Sequence of events:

1. The function increments the pointer by one.

2. The function copies data from input parameter IN to the position in the table indicated by the pointer. (It will write 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).

4. The pointer does not wrap around when the table is full.

The FIFOWRT function has four input parameters and two output parameters. When the function 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 the function does not pass power to the right. The pointer always indicates the last item entered into the table. If the table becomes full, it will not be possible to add more entries to it.

The FIFOWRT function passes power to the right after a successful execution.
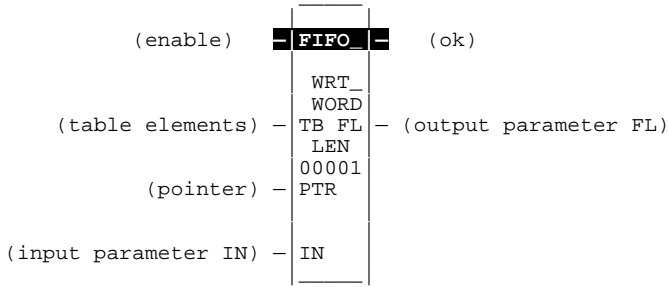
```
                              ┌──────┐
           (enable) ──┤ FIFO_ ├──   (ok)
                       │ WRT_  │
                       │ WORD  │
 (table elements) ──┤ TB FL ├── (output parameter FL)
                       │ LEN   │
                       │ 00001 │
          (pointer) ──┤ PTR   │
                       │       │
 (input parameter IN) ──┤ IN    │
                       └──────┘
```

Parameters:

Parameter	Description
enable	When enable is energized, the write operation is performed.
TB	TB contains the elements of the FIFO table.
PTR	PTR points to the last element of the FIFO table.
IN	IN contains the value to be written to the FIFO table.
ok	The ok output is energized when the function is enabled and PTR < LEN.
FL	Output FL is energized when the last element position of the FIFO table is written to.
LEN	LEN must be between 1 and 32,767.

Valid Memory Types:

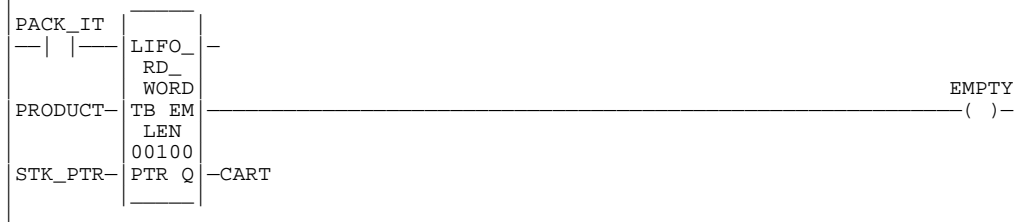| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
TB		o	o	o	o	Δ†	o		•	•	•	•	•	•		
PTR		•	•	•	•		•		•	•	•	•	•	•		
IN	•	o	o	o	o	Δ	o		•	•	•	•	•	•	•	
ok	•															•
FL	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, or WORD data only; not valid for DINT or DWORD.
Δ   Valid reference for WORD data only; not valid for INT, UINT, DINT, or DWORD.
†   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2..

Example:

In the following example, 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 will pass 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 FIFORD function.

```
   |        _____
   | UNPAC |     |
   —| |——| FIFO_ |—
   |      | WRT_  |
   |      | WORD  |                                                          FULL
   PRODUCT—| TB FL |————————————————————————————————————————————————————————( )—
   |      | LEN   |
   |      | 00100 |
   STK_PTR—| PTR   |
   |      |       |
   |      |       |
   P_CODE—| IN    |
   |      |_____|
   |
```

SORT   (INT, UINT, WORD)

The SORT function is used to sort an array in ascending order. The function has two input and two output parameters. EN is a Boolean enable; IN is the array to be sorted. OK is a Boolean output providing power flow, and Q is an array of integers that gives the index that the sorted elements had in the original array or list. Q is exactly the same size as IN. It also has a specification (LEN) of the number of elements to be sorted.

The SORT function is restricted to operate on arrays with up to 64 elements. When EN is ON, all of the elements of IN are sorted into ascending order, based on their 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.

Note

Do not use the SORT function in a timed or triggered input program block.

```
                           _____
                          |     |
   (enable)          ─█│ SORT_ │█─      (ok)
                          | WORD |
(input parameter IN) ─| IN  Q |─ (output parameter Q)
                          | LEN  |
                          |00001 |
                          |_____|
```
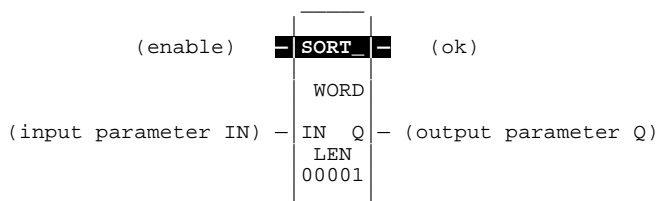
Parameters:

Parameter	Description
enable	When enable is energized, the sort is performed.
IN	IN contains the array of elements to be sorted. At the conclusion of the sort, the elements of IN are in sorted order.
ok	The ok output is energized whenever LEN is less than 65. However, the programmer does not limit the LEN parameter to 64. If LEN is greater than 64, the function block operates only on the first 64 units of LEN.
Q	Output Q contains an array of indexes that gives the position of the sorted elements in the original array.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN		•	•	•	•		•		•	•	•	•	•	•		
ok	•															•
Q	•	•	•	•	•		•		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following 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 will contain 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.

```
         |    _____
%Q00014  |   |     |
──| |────| LIFO_|─
         |   | WRT_|
         |   | UINT|                 _____
 PLIST ──| TB FL|───────────────────|     |                                    %Q00025
         |   | LEN |                 | SORT_|───────────────────────────────────( )─
         |   |00005|                 | UINT|
%L00041──| PTR |         PLIST ──| IN  Q|─PPOSN
         |   |     |                 | LEN |
         |   |     |                 |00005|
%I00017──| IN  |                 |_____|
         |   |_____|
         |
```
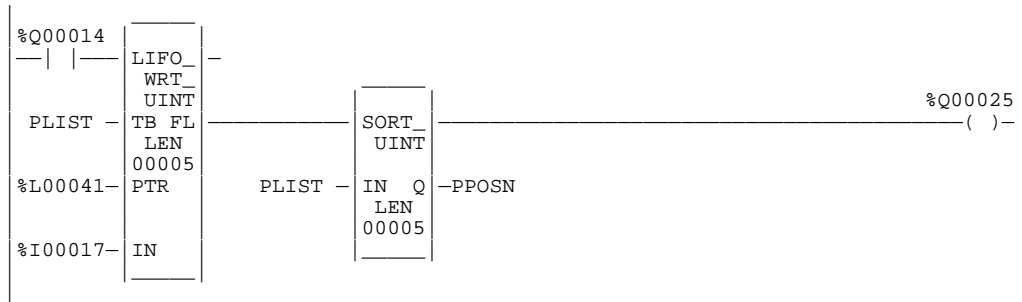
ARRAY_MOVE   (INT, UINT, DINT, BIT, BYTE, WORD, DWORD)

Use the Array Move (ARRAY_MOVE) function to copy a specified number of data elements from a source array to a destination array.

The ARRAY_MOVE function has six input parameters and two output parameters. When the function receives power flow, the number of data elements in the count indicator (N) is extracted from the input array starting with the indexed location $(SR + SNX – 1)$. The data elements are written to the output array starting with the indexed location $(DS + DNX – 1)$. The LEN operand specifies the number of elements that make up each array.

For ARRAY_MOVE_BIT, when word-oriented memory is selected for the parameters of the source array and/or destination array starting address, the least significant bit of the specified word is the first bit of the array. The value displayed contains 16 bits, regardless of the length of the array.

The indices in an ARRAY_MOVE instruction are 1-based. In using an ARRAY_MOVE, no element outside either the source or destination arrays, as specified by their starting address and length, may be referenced.

The ok output will receive power flow, unless one of the following conditions occurs:

- Enable is OFF.

- $(N + SNX)$ is greater than LEN.

- $(N + DNX)$ is greater than LEN.

```
                                        _____
            (enable)   ─ ■ ARRAY ■ ─   (ok)
                          MOVE_
                           BIT
(source array address) ─  SR DS ─ (destination array address)
                           LEN
                          00001
   (source array index) ─ SNX

(destination array index) ─ DNX

 (elements to transfer ) ─ N
                          _____
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
SR	SR contains the starting address of the source array. For ARRAY_MOVE_BIT, any reference may be used; it does not need to be byte aligned. However, 1 bit, beginning with the reference address specified, is displayed online.
SNX	SNX contains the index of the source array.
DNX	DNX contains the index of the destination array.
N	N provides a count indicator.
ok	The ok output is energized whenever the function is enabled. If NX is out of range, ok will not be energized.
DS	DS contains the starting address of the destination array. For ARRAY_MOVE_BIT, any reference may be used; it does not need to be byte aligned. However, 1 bit, beginning with the reference address specified, is displayed online.
LEN	LEN specifies the number of elements starting at SR and DS that make up each array.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
SR	•	o	o	o	o	Δ	o	o	•	•	•	•	•	•		
SNX	•	•	•	•	•		•		•	•	•	•	•	•	•	
DNX	•	•	•	•	•		•		•	•	•	•	•	•	•	
N	•	•	•	•	•		•		•	•	•	•	•	•	•	
DS	•	o	o	o	o	†	o		•	•	•	•	•	•		
ok	•															•

- •   Valid reference or place where power may flow through the function. For ARRAY_MOVE_MOVE_BIT, discrete user references %I, %Q, %M, and %T need not be aligned.
- o   Valid reference for INT, UINT, BIT, BYTE, or WORD data only; not valid for DINT or DWORD. %U is allowed for ARRAY_MOVE_BIT only.
- Δ   Valid data type for BIT, BYTE, or WORD data only; not valid for INT, UINT, DINT, or DWORD.
- †   %SA, %SB, %SC only; %S cannot be used.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2..
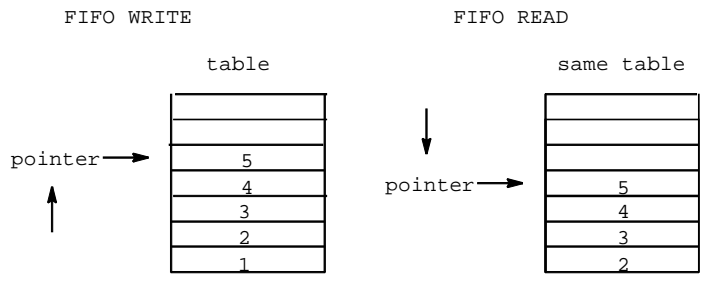
Example 1:

In this example, %R00003 - %R00007 of the array %R00001 - %R00016 is read and then written into %R00104 - %R00109 of the array %R00100 - %R00115.

```
           _____
 %I00001  |      |
 —| |—————| ARRAY |—
          | _MOVE |
          | _WORD |
 %R00001—| SR  DS |—%R00100
          |  LEN   |
          |  00016 |
 CONST  —| SNX    |
 00003    |        |
          |        |
 CONST  —| DNX    |
 00005    |        |
          |        |
 CONST  —| N      |
 00005    |_____|
```

Example 2:

Using bit memory for SR and DS, %M00011 - %M00017 of the array %M00009 - %M00024 is read and then written to %Q00026 - %Q00032 of the array %Q00022 - %Q00037.

```
           _____
 %I00001  |      |
 —| |—————| ARRAY |—
          | _MOVE |
          | _BIT  |
 %M00009—| SR  DS |—%Q00022
          |  LEN   |
          |  00016 |
 CONST  —| SNX    |
 00003    |        |
          |        |
 CONST  —| DNX    |
 00005    |        |
          |        |
 CONST  —| N      |
 00007    |_____|
```

Example 3:

Using word memory, for SR and DS, the third least significant bit of %R00001 through the second least significant bit of %R00002 of the array containing all 16 bits of %R00001 and four bits of %R00002 is read and then written into the fifth least significant bit of %R00100 through the fourth least significant bit of %R00101 of the array containing all 16 bits of %R00100 and four bits of %R00101.

```
              _____
  %I00001    |      |
  ──| |──────| ARRAY |─
             | _MOVE |
             | _BIT  |
  %R00001────| SR DS |─%R00100
             |  LEN  |
             | 00020 |
   CONST  ───| SNX   |
   00003      |      |
             |      |
   CONST  ───| DNX   |
   00005      |      |
             |      |
   CONST  ───| N     |
   00016     |_____|
```
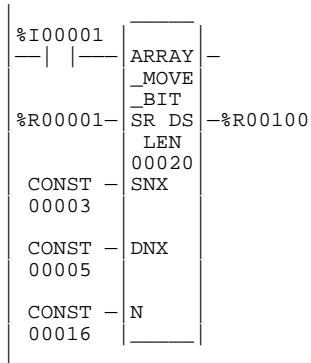
SRCH_EQ and SRCH_NE (INT, UINT, DINT, BYTE, WORD, DWORD)
SRCH_GT and SRCH_LT
SRCH_GE and SRCH_LE

Use the appropriate Search function listed below to search for all array values for that particular operation.

Abbreviation	Function	Description
SRCH_EQ	Search Equal	Search for all array values equal to a specified value.
SRCH_NE	Search Not Equal	Search for all array values not equal to a specified value.
SRCH_GT	Search Greater Than	Search for all array values greater than a specified value.
SRCH_GE	Search Greater Than or Equal	Search for all array values greater than or equal to a specified value.
SRCH_LT	Search Less Than	Search for all array values less than a specified value.
SRCH_LE	Search Less Than or Equal	Search for all array values less than or equal to a specified value.

Each function has four input parameters and two output parameters. When the function receives power, the array is searched starting at (AR + input NX). This is the starting address of the array (AR) plus the index into this array (input NX).

The search continues until the array element of the search object (IN) is found or until the end of the array is reached. If an array element is found, output parameter (FD) is set ON and output parameter (output NX) is set to the relative position of this element within the array. If no array element is found before the end of the array is reached, then output parameter (FD) is set OFF and output parameter (output NX) is set to zero.

The valid values for input NX are 0 to LEN – 1. NX should be set to zero to begin searching at the first element. This value increments by one at the time of execution. Therefore, the values of output NX are 1 to LEN. If the value of input NX is out-of-range, (< 0 or ≥ LEN), the value of output NX is set to the default value of zero.
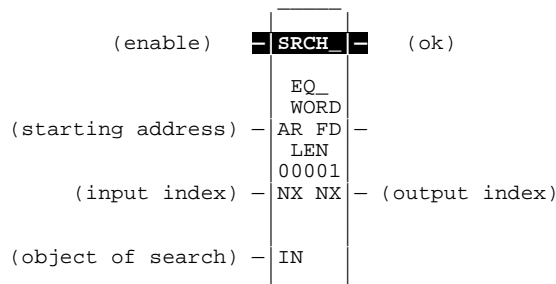
```
                                  _____
              (enable)  ─   ■ SRCH_ ■   ─   (ok)

                                   EQ_
                                   WORD
     (starting address)  ─  AR  FD  ─
                                   LEN
                                  00001
         (input index)  ─  NX  NX  ─  (output index)


      (object of search)  ─  IN
                                  _____
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
AR	AR contains the starting address of the array to be searched.
Input NX	Input NX contains the index into the array at which to begin the search.
IN	IN contains the object of the search.
Output NX	Output NX holds the position within the array of the search target.
FD	FD indicates that an array element has been found and the function was successful.
ok	The ok output is energized when the function is performed without error. If NX is out of range, ok will not be energized.
LEN	LEN specifies the number of elements starting at AR that make up the array to be searched. It may be 1 to 32,767 bytes or words.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
AR	•	o	o	o	o	Δ	o		•	•	•	•	•	•		
NX in	•	•	•	•	•		•		•	•	•	•	•	•	•	
IN	•	o	o	o	o	Δ	o		•	•	•	•	•	•	•	
NX out	•	•	•	•	•		•		•	•	•	•	•	•		
FD	•															•
ok	•															•

•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, BYTE, or WORD data only; not valid for DINT or DWORD.
Δ   Valid reference for BYTE,or WORD data only; not valid for INT, UINT, DINT, or DWORD.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example 1:

The array AR is defined as memory addresses %R00001 - %R00005. When EN is ON, the portion of the array between %R00004 and %R00005 is searched for an element whose value is equal to N. If %R00001 = 7, %R00002 = 9, %R00003 = 6, %R00004 = 7, %R00005 = 7, and %R00100 = 7, then the search will begin at %R00004 and conclude at %R00004 when FD is set ON and a 4 is written to %R00101.

```
  |        _____
  |%I00001 |      |
  |—| |——— | SRCH_|—
  |        | EQ_  |
  |        | INT  |                                                    %Q00001
  |%R00001—| AR FD|——————————————————————————————————————————————————————( )—
  |        | LEN  |
  |        | 00005|
  | CONST —| NX NX|—%R00101
  |  00003 |      |
  |        |      |
  |%R00100—| IN   |
  |        |_____|
  |
```
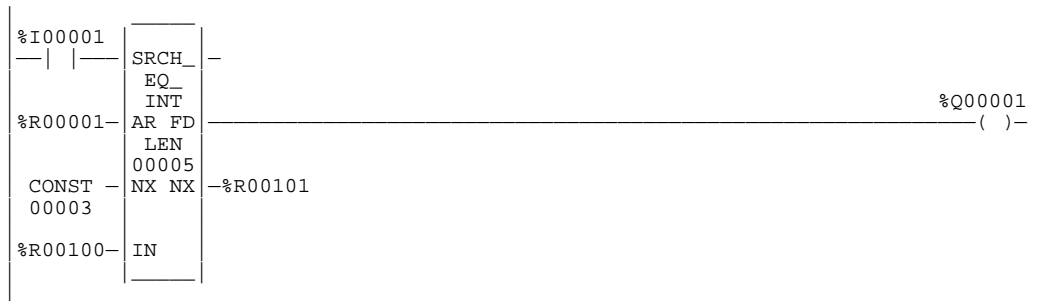
Example 2:

Array AR is defined as memory addresses %AI0001 - %AI0016. The values of the array elements are 100, 20, 0, 5, 90, 200, 0, 79, 102, 80, 24, 34, 987, 8, 0, and 500. Initially, %AQ0001 is 5. When EN is ON, each sweep will search the array looking for a match to the IN value of 0. The first sweep will start searching at %AI0006 and find a match at %AI0007, so FD is ON and %AQ0001 is 7. The second sweep will start searching at %AI0008 and find a match at %AI0015, so FD remains ON and %AQ0001 is 15. The next sweep will start at %AI0016. Since the end of the array is reached without a match, FD is set OFF and %AQ0001 is set to zero. The next sweep will start searching at the beginning of the array.

```
  |        _____
  |%I00001 |      |
  |—| |——— | SRCH_|—
  |        | EQ_  |
  |        | WORD |                                                    %M00001
  |%AI0001—| AR FD|——————————————————————————————————————————————————————( )—
  |        | LEN  |
  |        | 00016|
  |%AQ0001—| NX NX|—%AQ0001
  |        |      |
  |        |      |
  | CONST —| IN   |
  |  0000  |_____|
  |
```

ARRAY RANGE   (INT, DINT, WORD, DWORD)

The ARRAY RANGE function is used to compare 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 will be 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.

Note

The Array Range function is only available on a Release 5 or higher CPU.

The ARRAY RANGE function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
UINT	Unsigned integer.
WORD	Word data type.
DWORD	Double word data type.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, please refer to "Data Types" section on page 2-16.

When the function is enabled, the ARRAY RANGE function block will compare 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, the ok output will receive power flow.

```
         (enable) – | EQ_    |   (ok)
                    |        |
                    | INT    |
                    | LEN    |
  (lower limit) – | LL   Q | – (array of bits)
  (upper limit) – | UL     |
  (input value) – | IN     |
                    |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
LL	LL contains the lower limit of the range.
UL	UL contains the upper limit of the range.
IN	IN contains the value to be compared against each range specified by L1 and L2.
ok	The ok output is energized unless an error is encountered.
Q	Output Q is energized when the value in IN is within the range specified by L1 and L2, inclusive. **Note:** Q is not aligned. It is displayed in bit format. It will display either a 1 (ON) or a 0 (OFF) for the first array element. For discrete references, it represents the reference displayed. For word references, it represents the low order bit of the reference displayed.
LEN	LEN is the number of elements in the arrays.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
LL		o	o	o	o		o		•	•	•	•	•		•†	
UL		o	o	o	o		o		•	•	•	•	•		•†	
IN		o	o	o	o		o		•	•	•	•	•			
ok		o	o	o	o		o		•	•	•	•	•			
Q	•															•

<u>Note</u>: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•　　Valid reference or place where power may flow through the function.
o　　Valid reference for INT or WORD data only; not valid for DINT or DWORD.
†　　Constants are limited to integer values for double precision signed integer operations.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example 1:

In the following example, 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).

```
                 _____
 %I00001        |      |                                                         %Q00001
 —| |——————| ARRAY |—————————————————————————————————————————————————————————( )—
                | RANGE|
                |  INT |
                |      |
 %R00001—| LL  Q|—%R00200
                | LEN  |
                | 00008|
 %R00100—| UL   |
                |      |
      40—| IN   |
                |_____|
```

Example 2:

In the following example, 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 ok output will be set ON (1).

```
                 _____
 %I00001        |      |                                                         %M00001
 —| |——————| ARRAY |—————————————————————————————————————————————————————————( )—
                | RANGE|
                |  INT |
                |      |
 %T00001—| LL  Q|—%Q00001
                | LEN  |
                | 00003|
 %T00049—| UL   |
                |      |
      65—| IN   |
                |_____|
```

Chapter 11

Conversion Functions

Use the conversion functions to convert a data item from one number type to another. Many programming instructions, such as math functions, must be used with data of one type. This chapter describes the following conversion functions:

Abbreviation	Function	Description	Page
BCD-4	Convert to BCD-4	Convert an unsigned or signed integer to 4-digit BCD format.	11-2
BCD-8	Convert to BCD-8	Convert a double precision signed integer to 8-digit BCD format.	11-4
UINT	Convert to Unsigned Integer	Convert BCD-4, signed integer, or double precision signed integer to unsigned integer format.	11-6
INT	Convert to Signed Integer	Convert BCD-4, unsigned integer, or double precision signed integer to signed integer format.	11-8
DINT	Convert to Double Precision Signed Integer	Convert BCD-8, unsigned integer, or signed integer to double precision signed integer format.	11-10
REAL	Convert to Real	Convert BCD-4 BCD-8, unsigned integer, signed integer, or double precision signed integer to real value format.	11-12
TRUN	Truncate	Round the real number toward zero.	11-14

BCD-4   (INT, UINT)

The Convert to BCD-4 function is used to output the 4-digit BCD equivalent of unsigned or signed integer data. The original data is not changed by this function. The output data can be used directly as input for another program function.

Data can be converted to BCD format to drive BCD-encoded LED displays or presets to external devices, such as high-speed counters.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is outside the range 0 to 9999.

```
                          _____
                         |      |
   (enable)          ―― | INT_ | ――    (ok)
                         |      |
                         | TO_  |
                         | BCD4 |
(value to be converted) ―| IN  Q |― (output parameter Q)
                         |_____|
```
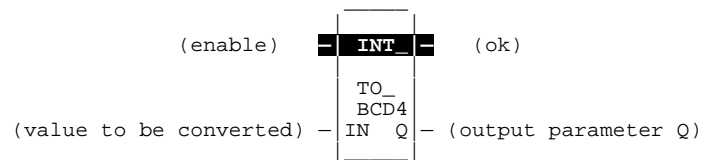
Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the integer value to be converted to BCD-4.
ok	The ok output is energized when the function is performed without error.
Q	Q contains the BCD-4 form of the original value in IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	•	•	•	•		•		•	•	•	•	•	•	•	
ok	•															•
Q	•	•	•	•	•		•		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function..

Note

For restrictions within a parameterized subroutine block, refer to the "Restrictions on Formal Parameters within a Parameterized Subroutine Block" section of Chapter 2.

Example:

In the following example, whenever input %I00002 is set and no errors exist, the integer at input location %I00017 through %I00032 is 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.

```
  |        _____
  |%I00002 |      |                                                          %M01432
  |—| |——— | INT_ |——————————————————————————————————————————————————————————( )—
  |        | TO_  |
  |        | BCD4 |
  |%I00017—| IN  Q|—%Q00033
  |        |_____|
  |
```
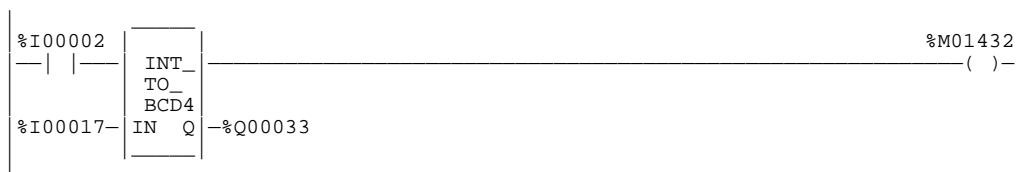
BCD-8   (DINT)

The Convert to BCD-8 function is used to output the 8-digit BCD equivalent of double precision signed integer data. The original data is not changed by this function. The output data can be used directly as input for another program function.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is outside the range 0 to 99999999.

```
                                   _____
                                  |      |
             (enable)           ─|▐DINT_|─     (ok)
                                  |      |
                                  | TO_  |
                                  | BCD8 |
  (value to be converted) ─|IN  Q|─ (output parameter Q)
                                  |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the integer value to be converted to BCD-8.
ok	The ok output is energized when the function is performed without error.
Q	Q contains the BCD-8 form of the original value in IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•								•	•	•	•	•	•	•	
ok	•															•
Q	•								•	•	•	•	•	•		

<u>Note</u>:   Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function..

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00002 is set and no errors exist, the double precision signed integer at input location %AI0003 is converted to eight BCD digits and the result is stored in memory locations %L00001 through %L00002.

```
  |        _____
  |%I00002 |      |
  |—| |——— |DINT_ |—
  |        | TO_  |
  |        | BCD8 |
  |%AI0003—|IN  Q |—%L00001
  |        |_____|
  |
```
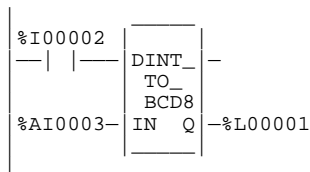
UINT (INT, DINT, BCD-4, REAL)

The Convert to Unsigned Integer function is used to output the integer equivalent of signed integer, double precision signed integer, BCD-4, or real data. The original data is not changed by this function. The output data can be used directly as input for another program function.

One use of the [3] UINT function 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.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is outside the range 0 to +65,535.
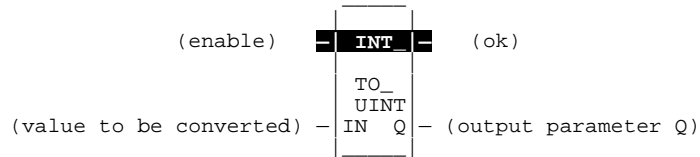
```
                                  _____
                                 |       |
        (enable)          ─| INT_ |─      (ok)
                                 |       |
                                 | TO_   |
                                 | UINT  |
(value to be converted) ─| IN  Q |─ (output parameter Q)
                                 |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the value to be converted to unsigned integer.
ok	The ok output is energized when the function is performed without error.
Q	Q contains the unsigned integer form of the original value in IN.

Valid Memory Types:

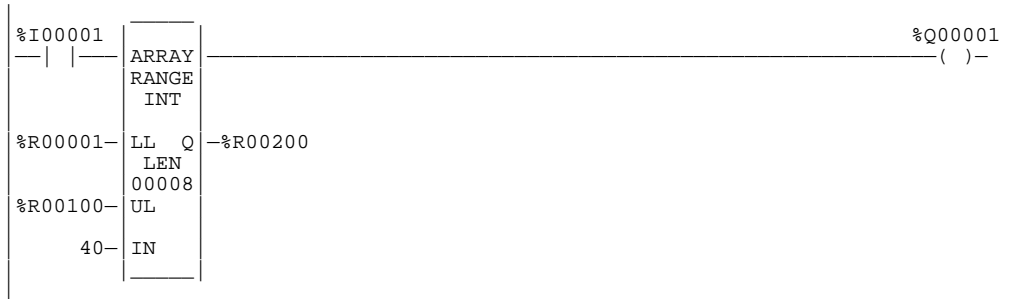| Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	•	•	•	•		•		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
• Valid reference or place where power may flow through the function.
o Not valid for DINT_TO_UINT..

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
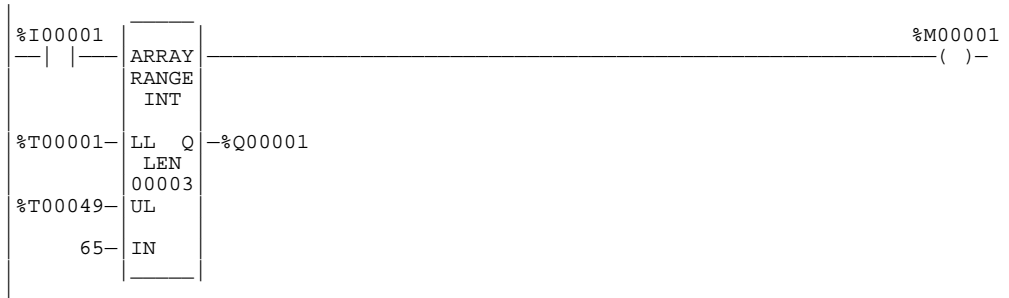
Example:

In the following example, whenever input %I00002 is set and no errors exist, the double precision signed integer at input location %R00007 is converted to an unsigned integer 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.

```
  |     _____
%I00002|      |              _____
—| |——| DINT_|—————————|SUB_ |—
  |    | TO_  |         |UINT |
  |    | UINT |         |     |
%R00007—| IN  Q|—————————|I1  Q|—%Q00033
  |    |_____|         |     |
  |           CONST  —|I2   |
  |           00145   |_____|
  |
```
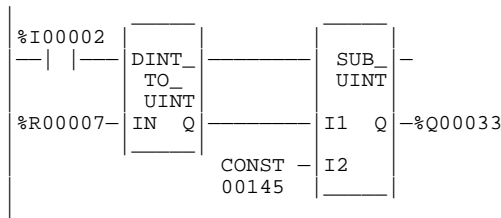
INT (UINT, DINT, BCD-4, REAL)

The Convert to Signed Integer function is used to output the integer equivalent of unsigned integer, double precision signed integer, or BCD-4 data. The original data is not changed by this function. The output data can be used directly as input for another program function.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function always passes power flow when power is received, unless the data is out of range.

```
                              _____
                             |      |
      (enable)          ─| UINT |─        (ok)
                             |      |
                             | TO_  |
                             | INT  |
(value to be converted) ─| IN  Q|─ (output parameter Q)
                             |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the value to be converted to integer.
ok	The ok output is energized whenever enable is energized, unless the data is out of range.
Q	Q contains the integer form of the original value in IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•	•	•	•	•		•		•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Not valid for DINT_TO_INT..

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, whenever input %I00002 is set, the BCD-4 value in PARTS is converted to a signed integer and passed to the ADD function, where it is added to the signed integer value represented by the reference RUNNING. The sum is output by the ADD function to the reference TOTAL.

```
    |
    | %I00002   _____            _____
    |  —| |—— |BCD—4 ———————  |ADD_ |—
    |          | TO_           | INT
    |          | INT           |
    |  PARTS —| IN  Q|———————  |I1  Q|—TOTAL
    |          |_____|         |
    |               RUNNING—| I2
    |                        |_____|
    |
```
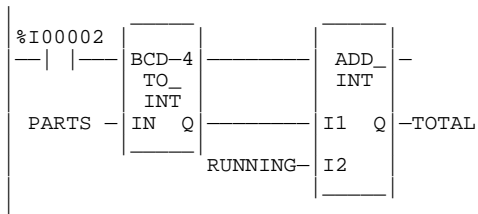
DINT   (INT, UINT, BCD-8, REAL)

The Convert to Double Precision Signed Integer function is used to output the double precision signed integer equivalent of unsigned integer, signed integer, or BCD-8 data. The original data is not changed by this function. The output data can be used directly as input for another program function.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function always passes power flow when power is received, unless the real value is out of range.

```
                                 _____
                                |      |
     (enable)            ─█  INT_  █─       (ok)
                                |      |
                                | TO_  |
                                | DINT |
(value to be converted) ─|IN  Q|─ (output parameter Q)
                                |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	In contains a reference for the value to be converted to double precision integer.
ok	The ok output is energized whenever enable is energized, unless the real value is out of range.
Q	Q contains the double precision signed integer form of the original value in IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•								•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Not valid for BCD8_TO_DINT..

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
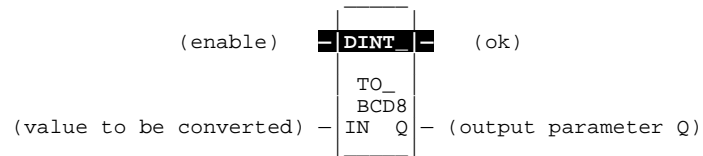
Example:

In the following example, whenever input %I00002 is set, the integer value at input location %I00017 is converted to a double precision signed integer and the result is placed in location %L00001. The output %Q01001 is set whenever the function executes successfully.

```
|    _____
|%I00002  |      |                                                    %Q01001
|—| |——| INT_ |————————————————————————————————————————————————————————( )—
|        | TO_  |
|        | DINT |
|%I00017—| IN  Q|—%L00001
|        |_____|
|
```
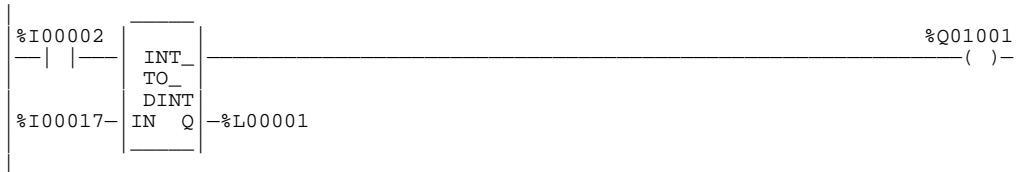
REAL   (INT, UINT, DINT, BCD-4, BCD-8)

The Convert to Real function is used to output the real value of the input data. The original data is not changed by this function. The output data can be used directly as input for another program function.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is out of range.

It is possible for a loss of precision to occur when converting from DINT or BCD-8 to REAL since the number of significant bits is reduced to 24.
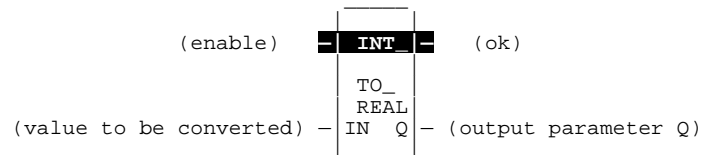
```
                                      _____
                                     |     |
         (enable)                 ──■| INT |■──    (ok)
                                     |     |
                                     | TO_ |
                                     | REAL|
(value to be converted)  ──────────|IN  Q|──  (output parameter Q)
                                     |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the integer value to be converted to REAL.
ok	The ok output is energized when the function is performed without error.
Q	Q contains the REAL form of the original value in IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•	o	o	o	o		o		•	•	•	•	•	•	•	
ok	•															•
Q	•								•	•	•	•	•	•		

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Not valid for DINT._TO_REAL or BCD8_TO_REAL..

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example:

In the following example, the integer value of input IN is 678. The result value placed in %T00016 is 678.000.

```
          |‾‾‾‾|
          |      |
——————————| INT_ |—
          | TO_  |
          | REAL |
%T00001——| IN  Q |—%T00016
          |_____|
```
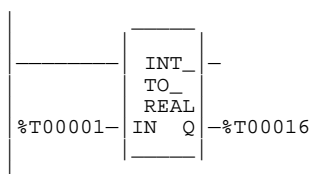
TRUN   (INT, DINT)

The Truncate function is used to round the real number toward zero. The original data is not changed by this function. The output data can be used directly as input for another program function.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes 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).
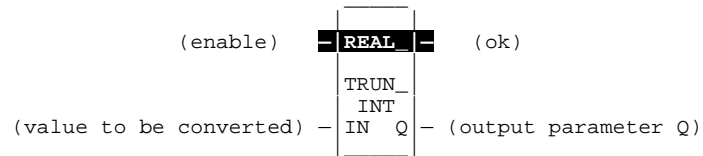
```
                             _____
                            |      |
            (enable)       ─| REAL |─    (ok)
                            |      |
                            |TRUN_ |
                            | INT  |
(value to be converted) ─  |IN   Q|─ (output parameter Q)
                            |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the real value to be truncated.
ok	The ok output is energized when the function is performed without error, unless the value is out of range or IN is NaN.
Q	Q contains the truncated INT or DINT value of the original value in IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN	•								•	•	•	•	•	•	•	
ok	•															•
Q	•	o	o	o	o		o		•	•	•	•	•	•		

<u>Note</u>: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Not valid for REAL_TRUN_INT only..

Note

> For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
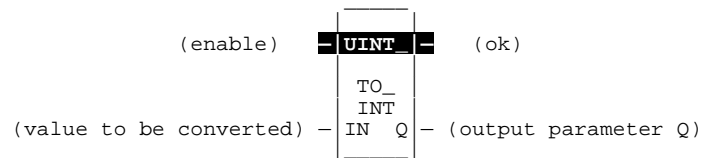
Example:

In the following example, the displayed constant is truncated and the integer result 562 is placed in %T00001.

```
|           _____
|          |     |
|──────────|REAL_|──
|          |TRUN_|
|          | INT |
|   CONST ─|IN  Q|─%T00001
|5.62987E+02|_____|
|
```
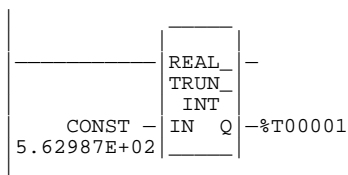
Chapter 12

Control Functions

This chapter describes the control functions, which may be used to limit program execution and change the way the CPU executes the application program. (Refer to chapter 2, section 1, "PLC Sweep Summary," for information on the CPU sweep.)

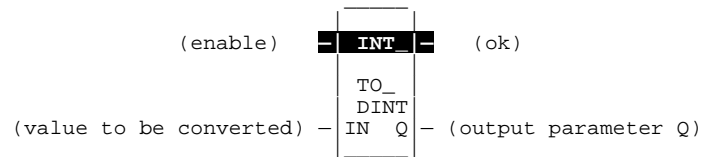| Function	Description	Page
CALL	Causes program execution to go to a specified program block.	12-3
CALL EXTERNAL	Causes program execution to go to a specified external block.	12-4
CALL SUBROUTINE	Causes program execution to go to a specified parameterized subroutine block.	12-6
DOIO	Services for one sweep a specified range of inputs or outputs immediately. (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 the real input points.	12-10
SUSIO	Suspends for one sweep all normal I/O updates, except those specified by DO I/O instructions.	12-14
MCR	Programs a Master Control Relay. An MCR causes all rungs between the MCR and its subsequent ENDMCR to be executed without power flow.	12-16
ENDMCR	Indicates that the subsequent logic is to be executed with normal power flow.	12-17
JUMP	Causes program execution to jump to a specified location (indicated by a LABEL, see below) in the logic.	12-18
LABEL	Specifies the target location of a JUMP instruction.	12-19
COMMENT	Places a comment (rung explanation) in the program. After entering the instruction, the text can be typed in by zooming into the instruction.	12-20
FOR, END_ FOR, EXIT	Repeat logic a specified number of times within a program.	12-21

Function	Description	Page
SVCREQ	Requests one of the following special PLC services:	12-25
	Change/Read Constant Sweep Timer.	12-28
	Read Window Values.	12-31
	Change Programmer Communications Window Mode and Timer Value.	12-32
	Change System Communications Window Mode and Timer Value.	12-33
	Change Background Task Window Mode and Timer Value.	12-34
	Change/Read Checksum Task State & No. of Words to Checksum.	12-36
	Change/Read Time-of-Day Clock State and Values.	12-38
	Reset Watchdog Timer.	12-42
	Read Sweep Time from Beginning of Sweep.	12-43
	Read Folder Name	12-44
	Read PLC ID.	12-45
	Read PLC Run State.	12-46
	Stop PLC.	12-47
	Clear Fault Tables.	12-48
	Read Last-Logged Fault Table Entry.	12-49
	Read Elapsed Time Clock.	12-53
	Mask/Unmask I/O Interrupt.	12-54
	Read I/O Override Status.	12-55
	Set Run Enable/Disable.	12-57
	Read Fault Tables.	12-58
	User-Defined Fault Logging.	12-62
	Mask/Unmask Timed Interrupts.	12-64
	Read Master Checksum.	12-65
	Disable/Enable EXE Block and Standalone C Program Checksums.	12-67
	Role Switch.	12-68
	Write to Reverse Transfer Area.	12-69
	Read from Reverse Transfer Area.	12-69
	Suspend/Resume I/O Interrupt	12-70
	ESCM Port Status	12-72
PID	Provides two PID (proportional/integral/derivative) closed-loop control algorithms: Standard ISA PID algorithm (PIDISA) and Independent term algorithm (PIDIND).	12-74

CALL

Use the CALL function to cause program execution to go to a specified program block. A CALL function can be used in any _MAIN program block, program block, or parameterized subroutine block .

Note

The maximum number of program block calls that can be programmed in a program block is 64. The maximum number of declared program blocks is 255. The maximum number of times you can call a subroutine is 255.

When the CALL function receives power flow, it causes the scan to go immediately to the designated program block and execute it. After the program block execution is complete, control returns to the point in the logic immediately following the CALL instruction.

CALL EXTERNAL

Use the CALL EXTERNAL function to cause program execution to go to a specified external block created outside the Logicmaster software. The eight CALL EXTERNAL functions vary in the number of parameters which convey input and output data blocks. The input and output parameters may have a length of one bit or one word. When a contact instruction adjoins one of the input or output parameters, the parameter is one bit in size. All data flow will be one bit in length. The enable and ok parameters are present for each function.

When the function is enabled, the specified external block is executed. The external logic operates on the blocks of incoming data and produces the blocks of output data. The ok parameter is controlled by logic within the external block.

After the program block execution is complete, control returns to the point in the logic immediately following the CALL instruction.

The CALL EXTERNAL function with no input and output parameters has the following form:

```
                          _____
(enable)        ─│CALL blkname│─    (ok)
                  │            │
                  │ (EXTERNAL) │
                  │_____│
```

The CALL EXTERNAL function may have up to seven pairs of parameters, as shown below:

```
                             _____
(enable)           ─│CALL blkname│─    (ok)
                     │            │
                     │ (EXTERNAL) │
(input parameter I1) ─│IN1      Q1│─ (output parameter Q1)

(input parameter I2) ─│IN2      Q2│─ (output parameter Q2)

(input parameter I3) ─│IN3      Q3│─ (output parameter Q3)

(input parameter I4) ─│IN4      Q4│─ (output parameter Q4)

(input parameter I5) ─│IN5      Q5│─ (output parameter Q5)

(input parameter I6) ─│IN6      Q6│─ (output parameter Q6)

(input parameter I7) ─│IN7      Q7│─ (output parameter Q7)
                     │_____│
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN1-IN7	The input parameters IN1 through IN7 indicate the starting location of data to be used within the external block.
ok	The ok output is controlled by logic within the external block.
Q1-Q7	The output parameters Q1 through Q7 indicate the starting location of data with output values.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
IN1-IN7	•	•	•	•		•	•	•	•	•	•	•	•	•	•	
ok	•															
Q1-Q7	•	•	•	•		•	•	•	•	•	•	•			•	•	

• Valid reference or place where power may flow through the function.

Example:

In the following example, if %I00001 is set, the external block named C_BLK20 is executed. Block C_BLK20 operates on the input data located at reference address %L00100 and produces values in the block of output data located at reference address %T00001. Logic within C_BLK20 controls the Q1 output.

```
     |  _____|
%I00001 |               |                                                      %Q01001
—| |——| CALL C_BLK20   |——————————————————————————————————————————————————————( )—
     |  (EXTERNAL)      |
     |                  |
%L00100—| IN1        Q1 |—%T00001
     |  _____|
     |
```
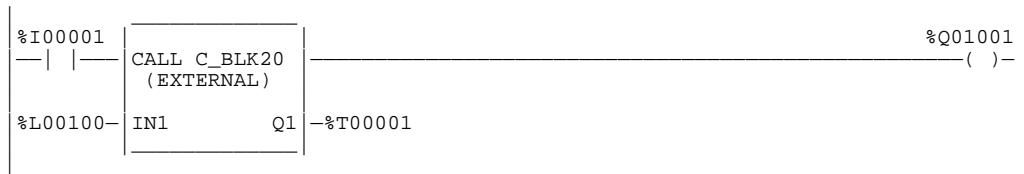
CALL SUBROUTINE

Use the CALL SUBROUTINE function to cause program execution to go to a specified parameterized subroutine. Parameterized subroutine blocks (PSBs) are user-defined function blocks, configured with between zero and seven input/output parameter pairs. A program can "call" a parameterized subroutine block as it executes; however, the parameterized subroutine block must be declared through the block declaration editor before a CALL instruction can be used for that parameterized subroutine block. For more information on declaring a parameterized subroutine block, refer to chapter 2, section 3, "Program Organization and User Data," in this manual and the *Logicmaster 90-70 Programming Software User's Manual*, GFK-0263.

When the function is enabled, the specified parameterized subroutine is executed. The logic in the subroutine operates on the blocks of incoming data and produces blocks of output data. The ok parameter may be controlled by logic within the subroutine. After the program block execution is complete, control returns to the point in the logic immediately following the CALL instruction.

The CALL SUBROUTINE function with no input or output parameters has this form:

```
                     _____
                    |              |
(enable)          ──| CALL blkname |──      (ok)
                    |              |
                    | (SUBROUTINE) |
                    |_____|
```

In this case of a zero-parameter PSB call, the ok parameter is always set to ON.

The CALL SUBROUTINE function may have up to seven pairs of parameters, as shown below:

```
                              _____
                             |              |
(enable)                   ──| CALL blkname |──      (ok)
                             |              |
                             | (SUBROUTINE) |
(input parameter X1)  ──|X1            Y1|── (output parameter Y1)

(input parameter X2)  ──|X2            Y2|── (output parameter Y2)

(input parameter X3)  ──|X3            Y3|── (output parameter Y3)

(input parameter X4)  ──|X4            Y4|── (output parameter Y4)

(input parameter X5)  ──|X5            Y5|── (output parameter Y5)

(input parameter X6)  ──|X6            Y6|── (output parameter Y6)

(input parameter X7)  ──|X7            Y7|── (output parameter Y7)
                         |_____|
```

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
X1-X7	The input parameters X1 through X7 indicate the starting location of data to be used within the subroutine block.
ok	The ok output may be controlled by logic within the subroutine block using PSB formal parameter Y0[001].
Y1-Y7	The output parameters Y1 through Y7 indicate the starting location of the output data.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
X1-X7	•†	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ok	•															
Y1-Y7	•‡	•	•	•	•		•	•	•	•	•	•	•	•		•

•   Valid reference or place where power may flow through the function.
†   Bit parameters only (when bit length is 1).
‡   When word length is 1.

Note

PSB (Parameterized Subroutine Block) formal BIT parameters are **not** allowed as parameters to the CALL SUBROUTINE function. PSB formal WORD and NWORD parameters are allowed as parameters to the CALL SUBROUTINE function. For restrictions within a parameterized subroutine block, refer to the "Restrictions on Formal Parameters within a Parameterized Subroutine Block" section of Chapter 2.

Note

Indirect references may be allowed as assigned input parameters, but not as formal parameters. %S references, like ALW_ON and ALW_OFF, may be required in some applications to take the place of required but unused assigned input parameters. %S references may not be assigned to WORD type input parameters, but may be assigned to BIT type input parameters.

Example:

In the following example, if %I00001 is set, the parameterized subroutine block named BLK20 is executed. Block BLK20 operates on the input data located at reference addresses %L00100 – %L00109 and produces values in the block of output data located at reference addresses %T00001 – %T00016. Logic within BLK20 controls the Y1 output.

```
         _____
%I00001 |                |                                                         %Q01001
—| |——  |  CALL BLK20    |————————————————————————————————————————————————————————( )—
        | (SUBROUTINE)   |
        |                |
        | W010      B016 |
%L00100—|X1          Y1 |—%T00001
        |                |
        |_____|
```
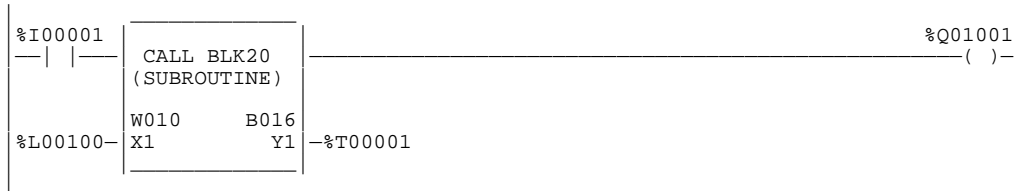
Parameterized Subroutine Block Example

Parameterized subroutine blocks are useful for building libraries of user-defined functions that can be used and moved from one application to another. For example, if you had a mathematical equation or algorithm that needed to be performed several times, you could write a parameterized subroutine block for that function and then call it as needed throughout your program.

For example, if you have a mathematical equation such as:

$$E = (A + B + C + D) / 4$$

where E is a word output, and A, B, C, and D are word inputs. A parameterized subroutine named AVG_4 could be defined and called as follows:

```
         _____
%I00001 |                |
—| |——  |  CALL AVG_4    |
        | (SUBROUTINE)   |
        |                |
        | W001      W001 |
%R00001—| A            E |—%R00005
        |                |
        | W001      NONE |
%R00002—| B           Y2 |—
        |                |
        | W001      NONE |
%R00003—| C           Y3 |—
        |                |
        | W001      NONE |
%R00004—| D           Y4 |—
        |                |
        |_____|
```
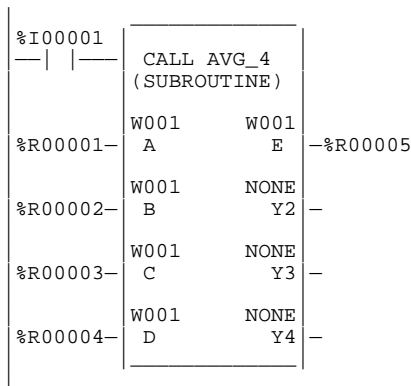
In this example, the average of %R00001, %R00002, %R00003, and %R00004 would be placed in %R00005.

The logic within the parameterized subroutine block would then look like this:

```
   |
   |                ____                   ____                   ____
   |               | ADD_ |               | ADD_ |               | DIV_ |
   |───────────────| UINT |───────────────| UINT |───────────────| UNIT |
   |               |      |               |      |               |      |
   A[001]─|I1   Q|───────────|I1   Q|───────────|I1   Q|─E[001]
   |               |      |               |      |               |      |
   B[001]─|I2    |      +─|I2    |      0004─|I2    |
   |               |_____|               |_____|               |_____|
   |
   |
   |                ____
   |               | ADD_ |
   |───────────────| UINT |
   |               |      |
   C[001]─|I1   Q|───────────+
   |               |      |
   D[001]─|I2    |
   |               |_____|
   |
```
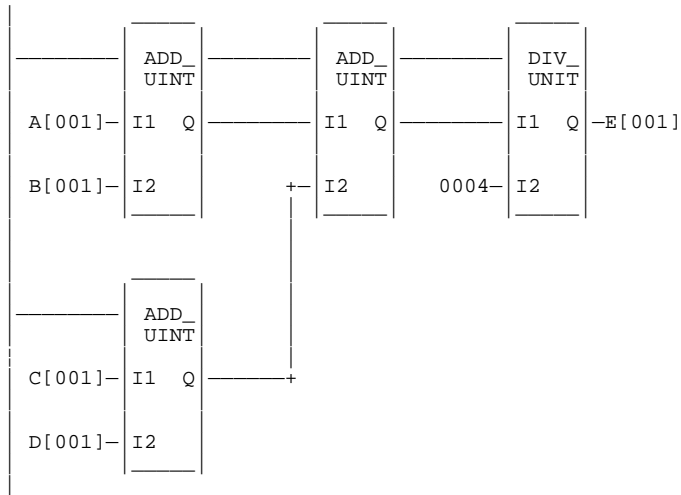
DOIO

The DO I/O (DOIO) function is used to update inputs or outputs for one scan while the program is running. The DOIO function can be used in conjunction with a SUSIO function, which stops the normal I/O scan. It can also be used to update selected I/O during the program in addition to the normal I/O scan.

If input references are specified, the function 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 points are serviced in increments of entire I/O modules; the PLC adjusts the references, if necessary, while the function executes. The DOIO function will not scan I/O modules that are not configured.

Note

The DOIO function is supported for Series 90-70 I/O modules only. It does not support Genius I/O modules and does not affect Ethernet Global Data production or Consumption. For details, refer to Chapter 4 in the *TCP/IP Ethernet Communications for the Series 90 PLC User's Manual*, GFK-1541.

The DOIO function has four input parameters and one output parameter. When the function receives power flow and input references are specified, the input points at the starting reference ST and ending at END are scanned. 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 points are not updated. ALT must be the same size as the reference type scanned. If a discrete reference is used for ST and END, then ALT must also be discrete. If no reference is specified for ALT, the real input points are updated.

When the DOIO function receives power flow and output references are specified, the output points at the starting reference ST and ending at END are written to the output modules. If outputs should be written to the output modules from internal memory, other than %Q or %AQ, the beginning reference can be specified for ALT. The range of outputs written to the output modules is specified by the starting reference ST and the ending reference END.

Execution of the function continues until either all inputs in the selected range have reported, or all outputs have been serviced on the I/O cards. Program execution then returns to the next function following the DO I/O.

The function passes power to the right whenever power is received, 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 Module" fault.

Note

If the DOIO function is used with timed or I/O interrupts, transitional contacts associated with scanned inputs may not operate as expected.

```
                             _____
                            |        |
              (enable)  ─|█DO_IO█|─  (ok)
                            |        |
    (starting address) ─| ST     |
                            |        |
      (ending address) ─| END    |
                            |        |
                        ─| ALT    |
                            |_____|
```
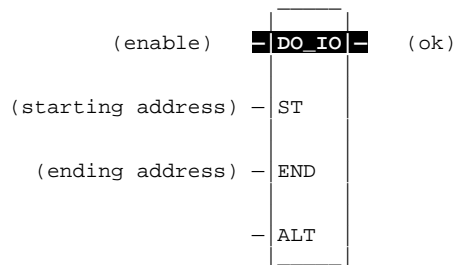
Parameters:

Parameter	Description
enable	When the function is enabled, a limited input or output scan is performed.
ST	ST is the starting address of the set of input or output points or words to be serviced.
END	END is the ending address of the set of input or output points or words to be serviced.
ALT	For the input scan, ALT specifies the address to store scanned input point/word values. For the output scan, ALT specifies the address to get output point/word values from to send to the I/O modules.
ok	The ok output is energized when the input or output scan completes normally.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
ST		•	•									•	•	•		
END		•	•									•	•	•		
ALT		•	•	•	•		•		•			•	•	•		•
ok	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
•   Valid reference or place where power may flow through the function.
o   Valid reference for INT, UINT, or WORD data only; not valid for DINT or DWORD.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Input Example 1:

In the following example, when the enabling input %I00001 is ON, references %I00001 through %I00064 are scanned and %Q00001 is turned on. A copy of the scanned inputs is placed in internal memory from reference %M00001 through %M00064. The real input points are not updated. This form of the function can be used to compare the current values of input points with the values of input points at the beginning of the scan.

```
         _____
%I00001 |      |                                                      %Q00001
——| |——| DO_IO |——————————————————————————————————————————————————————( )—
        |      |
%I00001—|ST    |
        |      |
%I00064—|END   |
        |      |
%M00001—|ALT   |
        |_____|
```
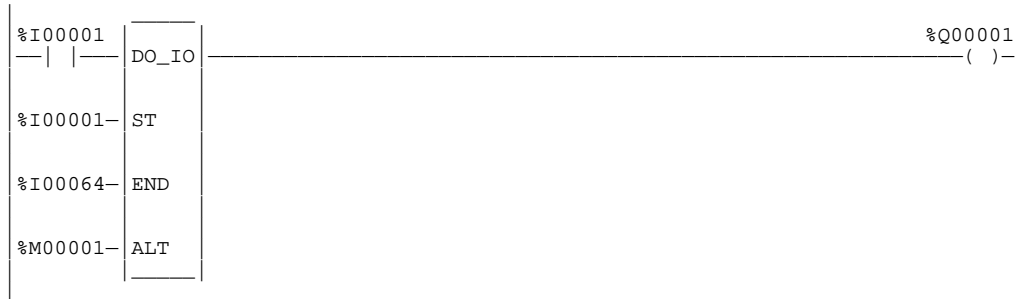
Input Example 2:

In the following example, when the enabling input %I00001 is ON, references %I00001 through %I00064 are scanned and %Q00001 is turned on. The scanned inputs are placed in the input status memory from reference %I00001 to %I00064. This form of the function allows input points to be scanned one or more times during the program execution portion of the CPU sweep.

```
         _____
%I00001 |      |                                                      %Q00001
——| |——| DO_IO |——————————————————————————————————————————————————————( )—
        |      |
%I00001—|ST    |
        |      |
%I00064—|END   |
        |      |
      —|ALT    |
        |_____|
```
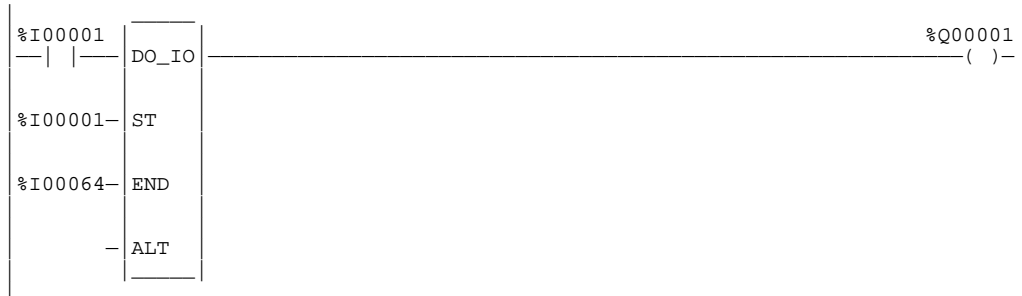
Output Example 1:

In the following example, when the enabling input %I00001 is ON, the values at references %R00001 through %R00004 are written to analog output channels %AQ0001 through %AQ0004 and %Q00001 is turned on. The values at %AQ0001 through %AQ0004 are not written to the analog output modules.

```
           _____
%I00001   |      |                                                      %Q00001
——| |————|DO_IO |——————————————————————————————————————————————————————( )—
          |      |
%I00001——|ST    |
          |      |
%I00064——|END   |
          |      |
%M00001——|ALT   |
          |_____|
```
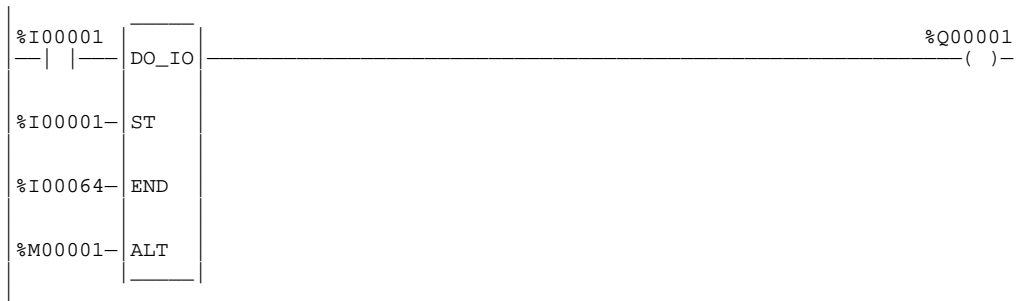
Output Example 2:

In the following example, when the enabling input %I00001 is ON, the values at references %AQ0001 through %AQ0004 are written to analog output channels %AQ0001 through %AQ0004 and %Q00001 is turned on.

```
           _____
%I00001   |      |                                                      %Q00001
——| |————|DO_IO |——————————————————————————————————————————————————————( )—
          |      |
%I00001——|ST    |
          |      |
%I00064——|END   |
          |      |
        —|ALT   |
          |_____|
```
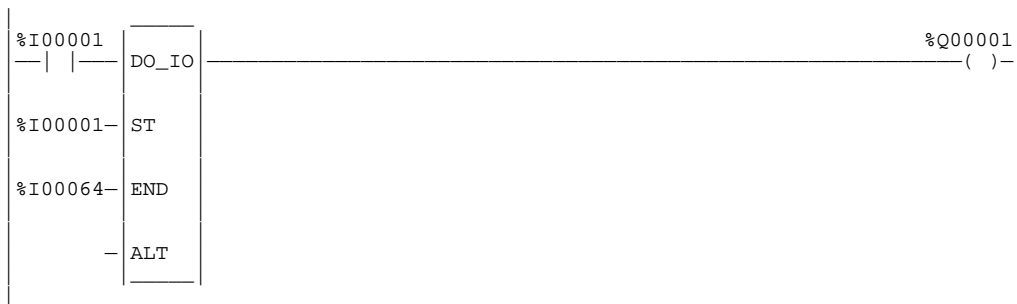
SUSIO

The Suspend I/O (SUSIO) function is used to stop 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 will verify that Genius bus controllers have completed their previous output updates.

Note

The SUSIO function suspends all I/O, both analog and discrete, whether integrated I/O, Genius I/O, or Ethernet Global Data. For details, refer to Chapter 4 in the *TCP/IP Ethernet Communications for the Series 90 PLC User's Manual*, GFK-1541.

The SUSIO function has one input and one output. When the function receives power flow, all I/O servicing stops except that provided by DOIO functions. If the function 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.

The SUSIO function passes power flow to the right whenever power is received.

```
 |   _____
 |─| SUSIO |─
 |   ‾‾‾‾‾‾
```

Example:

This example shows a SUSIO function and a DOIO 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 SUSIO function active on each sweep until %I00011 goes on. If this input were not scanned by the DO after the SUSIO went active, the SUSIO could only be disabled by powering down the PLC.

Output %Q00002 is set when both DOIO functions execute successfully. The rung is constructed so that both DOIO functions will execute even if one does not set its OK output. With normal I/O suspended, output %Q00002 is not updated until a DOIO function with %Q00002 in its range executes. This will not occur until the sweep after the setting of %Q00002. Outputs that are set after a DOIO function executes are not updated until another DOIO function executes, typically in the next sweep. Because of this delay, most programs that use SUSIO and DOIO place the SUSIO function in the first rung of the program, the DOIO function that processes inputs in the next rung, and the DOIO function that processes outputs in the last rung.

The range of the DOIO function doing outputs is %Q00001 through %Q00030. If the module in this range were a 32-point module, the DOIO function would actually perform a scan of the entire module. A DOIO function will not break the scan in the middle of an I/O module.

```
 |
 | [ START OF PROGRAM ]
 |
 |%I00010 %I00011  _____                                            %M00001
 |—| |—+—|/|——|SUSIO|————————————————————————————————————————————————( )—
 |        |        ‾‾‾‾‾‾
 |%M00001 |
 |—| |—+
 |                _____
 |%M00001         |                                                  %M00551
 |—| |————|DO_IO |————————————————————————————————————————————————( )—
 |        |      |
 |        |      |
 |%I00001—|ST    |
 |        |      |
 |%I00016—|END   |
 |        |      |
 |        |ALT   |
 |        |_____|
 |
 .
 .
 .
 |                _____
 |%M00001         |      | %M00551                                   %Q00002
 |—| |————|DO_IO |—| |————————————————————————————————————————( )—
 |        |      |
 |%Q00001—|ST    |
 |        |      |
 |%Q00030—|END   |
 |        |      |
 |        |ALT   |
 |        |_____|
 |
 | [ END OF PROGRAM ]
 |
```
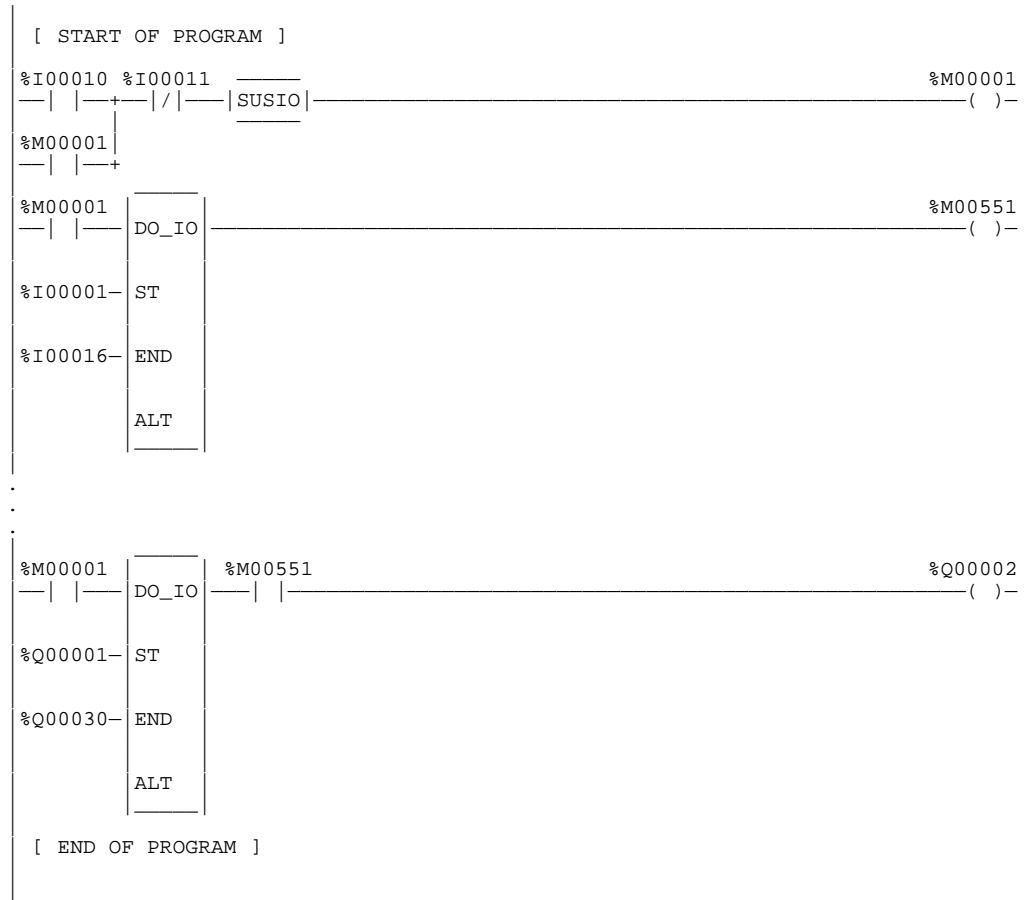
MCR

All rungs between an active Master Control Relay (MCR) and its corresponding End Master Control Relay (ENDMCR) are executed with negative logic. Use the MCR function to cause a portion of the program logic to be bypassed. An ENDMCR function associated with the MCR is used to resume normal program execution. Unlike the JUMP instruction, MCRs can only occur in the forward direction. The ENDMCR instruction must appear after its corresponding MCR instruction in a program.

Note

Program block calls within the scope of an active MCR will not execute.
However, any timers in the program block will continue to accumulate time.

There can be only one MCR instruction for each ENDMCR instruction. An MCR/ENDMCR pair can be nested within other MCR/ENDMCR pairs.

The MCR function has an enable Boolean input (EN) and also a name which identifies the MCR. This name is used again with an ENDMCR instruction. The MCR function has no outputs; there can be nothing after an MCR in a rung.

```
─[ MCR ]─
```

Example:

In the following example, whenever %I00002 allows power flow into the MCR function, program execution will continue without power flow to the coils until the associated ENDMCR is reached. If %I00001 and %I00003 are ON, %Q00001 is turned OFF and %Q00003 remains ON.

```
 |
 | %I0002    FIRST
 |─| |──[ MCR ]
||
||
|| %I00001                                                              %Q0001
|─| |──────────────────────────────────────────────────────────────────( )─
||
||
|| %I0003                                                               %Q0003
|─| |──────────────────────────────────────────────────────────────────(S)─
||
||
 |  FIRST
+[    ENDMCR    ]
 |
```
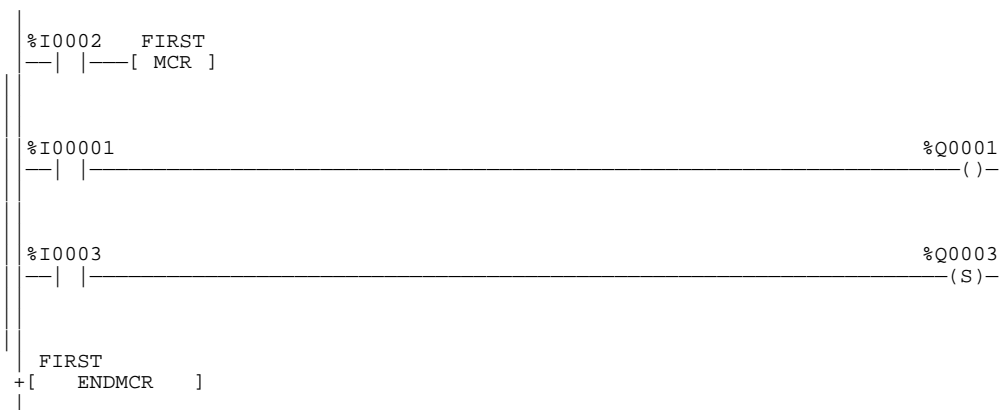
ENDMCR

Use the End Master Control Relay (ENDMCR) function to resume normal program execution after an MCR function. When the MCR associated with the ENDMCR is active, the ENDMCR causes program execution to resume with normal power flow. When the MCR associated with the ENDMCR is not active, the ENDMCR has no effect.

The ENDMCR function has a negated Boolean input (EN). The instruction enable must be provided by the power rail; execution cannot be conditional. The ENDMCR function also has a name, which identifies the ENDMCR and associates it with the corresponding MCR(s). The ENDMCR function has no outputs; there can be nothing before or after an ENDMCR instruction in a rung.

```
[   END MCR ]-
```

Example:

In the following example, an ENDMCR instruction is programmed to terminate MCR range "clear."

```
|
|   CLEAR
|—[      END MCR ]
|
```

JUMP

Use the JUMP instruction to cause a portion of the program logic to be bypassed. The JUMP can be either a forward or a backward JUMP. Program execution will continue at the LABEL specified.

The JUMP instruction is similar to an MCR, except that coils within the range of the JUMP are not executed with negative logic. When the JUMP is active, all coils within its scope are frozen. This includes coils associated with timers, counters, latches, and relays.

The JUMP instruction is always placed in columns 9 and 10 of the current rung line; there can be nothing after the JUMP instruction in the rung. Power flow jumps directly from the instruction to the rung with the named LABEL.

```
─────────────────────────────────────────────────────────━━━━━━━━━>> ???????
```

```
┌─────────────────┐
│     Caution     │
└─────────────────┘
```

To avoid creating and endless loop with forward and backward JUMP instructions, a backward JUMP must contain a way to make it conditional.

Example:

In the following example, whenever JUMP TEST1 is active, program execution is transferred to LABEL TEST1.

```
│
│%I00001
├─| |───────────────────────────────────────────────────────────────>> TEST1
│
```

LABEL

The LABEL instruction functions as the target destination of a JUMP. Use the LABEL instruction to resume normal program execution after a JUMP instruction.

There can be only one LABEL with a particular label name in a program. Programs without a matched JUMP/LABEL pair can be created and stored to the PLC, but cannot be executed.

The LABEL instruction has no input parameters and no output parameters; there can be nothing either before or after a LABEL in a rung.

Example:

In the following example, when JUMP TEST1 receives power, program execution is transferred to LABEL TEST1.

```
 TEST1 :
```

COMMENT

Use the COMMENT function to enter a comment (rung explanation) in the program. A comment can have up to 2048 characters of text. It is represented in the ladder logic like this:

```
|
|(*   COMMENT   *)
|
```

The text can be read or edited by moving the cursor to **(*   COMMENT   *)** after accepting the rung and selecting Zoom (F10). Comment text can also be printed.

Longer text can be included in printouts using an annotation text file, as described below:

1. Create the comment:

    A. Enter text to the point where the text from the other file should begin.

    B. Move the cursor to the beginning of a new line and enter **\I** or **\i**, the drive followed by a colon, the subdirectory or folder, and the file name, as shown in this example:

        \I d:\text\commnt1

       If the file is located on the same drive as the program folder, the drive designation is not necessary.

    C. Continue editing the program, or exit to MS-DOS.

2. After exiting the programmer, create a text file using any MS-DOS compatible software package. Give the file the file name entered in the comment, and place it on the drive specified in the comment.

FOR, END_FOR, and EXIT

FOR, END_FOR, and EXIT instructions enable you to repeat rung logic a specified number of times while varying the value of the FOR INDEX_VAR in the loop. The rung logic to be repeated must be placed between the FOR and END_FOR instructions.

The FOR instruction has five input parameters; there are no out parameters. When there is power flow at enable EN of the FOR instruction, the START, END, and INCREMENT parameters are saved. These saved values are used to evaluate the number of times the rungs between the FOR and its END_FOR instructions are executed. Changing the START and END parameters while the FOR loop is executing will not change its operation.

When there is power flow into the EXIT instruction, the FOR instruction is terminated and power flow jumps directly to the statement following the END_FOR instruction.

There can be nothing after the FOR instruction in the rung. An EXIT statement can only be placed between a FOR instruction and an END_FOR instruction in a program, and there can be nothing after the EXIT statement in the rung, as well. The END_FOR statement must be the only instruction in the rung.

```
                        _____
                       |  FOR_LOOP |
                       |           |
                     __| EN        |
                       |           |
            ???????__| INDEX_VAR   |
                       |           |
            ???????__| START       |
                       |           |
            ???????__| END         |
                       |           |
            ???????__| INCREMENT   |
                       |_____|


                        _____
                     __|  EXIT_FOR |
                       |_____|


                        _____
                     __|  END_FOR  |
                       |_____|
```
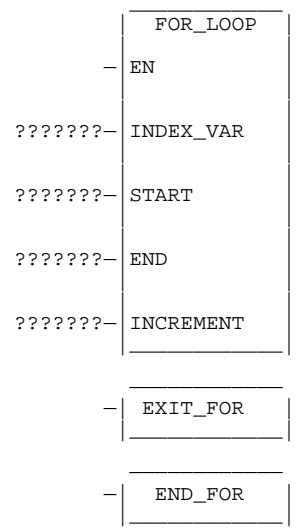
A FOR_LOOP can assign decreasing values to its index variable by setting the increment to a negative number. If the START value is 21, the END value is 1, and the increment value is –5, the statements of the FOR loop are executed five times, and the index variable is decremented by 5 in each pass. The 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 to reach the END or START cannot be decremented to reach the END, the statements within the FOR loop are not executed. For example, the value of START is 10, the value of END is 5, and the INCREMENT is 1. Power flow will jump directly from the FOR statement to the statement after the END_FOR statement.

Note

If the EN parameter for 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 will occur 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/END_FOR pair. JUMPs and LABEL instructions must also be completely within or completely outside the scope of a FOR/END_FOR pair. Jumping into or out of the scope of a FOR/END_FOR is not allowed.

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
INDEX_VAR	INDEX_VAR contains the index variable. When the loop has completed, this value is undefined. Changing the value of the index variable within the scope of the FOR loop is not recommended.
START	START contains the index start value.
END	END contains the index end value.
INCREMENT	INCREMENT contains the increment value.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
INDEX		•	•	•	•		•		•	•	•	•	•			
START	•	•	•	•	•		•		•	•	•	•	•	•	•	
END	•	•	•	•	•		•		•	•	•	•	•	•	•	
INCREMENT															•	•

• Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.

Example 1:

In the first example, the value for %M00001 (START) is 1, and the value for %M00017 (END) is 10. The INDEX_VAR (%R00001) will increment by the INCREMENT (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 I1 (%R00001) 1 ... 10 to I2 (%R00002).

```
|
|             _____
|            |  FOR_LOOP  |
|%I00001     |            |
|—| |——      | EN         |
|            |            |
|            |            |
|%R00001—    | INDEX_VAR  |
|            |            |
|            |            |
|%M00001—    | START      |
|            |            |
|            |            |
|%M00017—    | END        |
|            |            |
|            |            |
|        —   | INCREMENT  |
|            |_____|
|             _____
|%I00001     |      |
|—| |——      | ADD_ | —
|            |  INT |
|            |      |
|%R00001—    | I1  Q| —%R00003
|            |      |
|            |      |
|%R00002—    | I2   |
|            |_____|
|             _____
|—|     END_FOR      |
|            |_____|
|
|
```

Example 2:

In this next example, the value for %T00001 (START) is ,100, and the value for %T00017 (END) is 100. The INDEX_VAR (%R00001) will increment by tens, starting at ,100 until it reaches it end value of +100. The EQ function of the loop will try to execute 21 times, with I1 (%R00001) equal to –100, –90, –80, –70, –60, –50, –40, –30, –20, –10, 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. When I1 (%R00001) is 0, the EXIT statement will be enabled and power flow will jump directly to the statement after the END_FOR statement.

```
|
|              _____
|             |  FOR_LOOP |
| %I00001     |           |
|——| |————————| EN        |
|             |           |
|             |           |
| %R00001—————| INDEX_VAR |
|             |           |
|             |           |
| %T00001—————| START     |
|             |           |
|             |           |
| %T00017—————| END       |
|             |           |
|  CONST —————| INCREMENT |
|  +00010     |_____|
|             _____
| %I00001    |           |
|——| |———————| EQ_    —  |
|            |   INT     |
|            |           |          _____
| %R00001————| I1   Q|————————————|  EXIT_FOR   |
|            |           |         |_____|
|  CONST ————| I2        |
|  +00000    |_____|
|             _____
|——|   END_FOR   |
|            |_____|
|
```

SVCREQ

Use the Service Request (SVCREQ) function to request one of the following special PLC services:

Table 12-1. Service Request Functions

Function	Description	Page
1	Change/Read Constant Sweep Timer.	12-28
2	Read Window Values.	12-31
3	Change Programmer Communications Window Mode and Timer Value.	12-32
4	Change System Communications Window Mode and Timer Value.	12-33
5	Change Background Task Window Mode and Timer Value.	12-34
6	Change/Read Checksum Task State and Number of Words to Checksum.	12-36
7	Change/Read Time-of-Day Clock State and Values.	12-38
8	Reset Watchdog Timer.	12-42
9	Read Sweep Time from Beginning of Sweep.	12-43
10	Read Folder Name.	12-44
11	Read PLC ID.	12-45
12	Read PLC Run State.	12-46
13	Shut Down PLC.	12-47
14	Clear Fault Tables.	12-48
15	Read Last-Logged Fault Table Entry.	12-49
16	Read Elapsed Time Clock.	12-53
17	Mask/Unmask I/O Interrupt.	12-54
18	Read I/O Override Status.	12-55
19	Set Run Enable/Disable.	12-57
20	Read Fault Tables.	12-58
21	User-Defined Fault Logging.	12-62
22	Mask/Unmask Timed Interrupts.	12-64
23	Read Master Checksum.	12-65
24	Reserved.	–
25	Disable/Enable EXE Block and Standalone C Program Checksums.	12-67
26	Role Switch	12-68
27	Write to Reverse Transfer Area.	12-69
28	Read from Reverse Transfer Area.	12-69
29–31	Reserved.	–
32	Suspend/Resume I/O Interrupt.	12-70
39	ESCM Port Status	12-72
44	Logic Driven Dynamic Ethernet Global Data	12-74

The SVCREQ function has three input parameters and one output parameter. When the SVCREQ receives power flow, the PLC is requested to perform the function (FNC) indicated. Parameters for the function begin at the reference given for PARM. The SVCREQ function passes power flow

unless an incorrect function number, incorrect parameters, or out-of-range references are specified. Additional causes for failure are described on the pages that follow.

The reference given for PARM is the first of a group that make up the "parameter block" for the function. Successive 16-bit locations store additional parameters. The total number of references required will depend on the type of SVCREQ function being used.

Parameter blocks may be used as both inputs for the function and the location where data may be output after the function executes. Therefore, data returned by the function is accessed at the same location specified for PARM.

```
                              _____
                             |      |
(enable)  ──┤  SVC_  ├──  (ok)
                             |      |
                             | REQ  |
                             |      |
(service number) ──│FNC      |
                             |      |
(beginning reference) ──│PARM |
                             |_____|
```
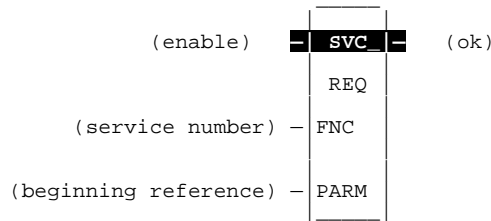
Parameters:

Parameter	Description
enable	When enable is energized, the requested service is performed.
FNC	FNC contains the constant or reference for the requested service.
PARM	PARM contains the beginning reference for the parameter block for the requested service.
ok	The ok output is energized when the function is performed without error.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%U	%R	%P	%L	%AI	%AQ	%UR	const	none
enable	•															
FNC		•	•	•	•		•		•	•	•	•	•	•	•	
PARM		•	•	•	•		•		•	•	•	•	•			
ok	•															•

Note: Indirect referencing is available for all register references (%R, %P, %L, %AI, %AQ, and %UR).
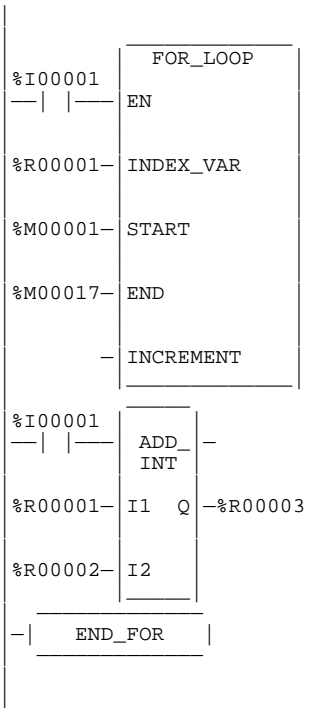• Valid reference or place where power may flow through the function.

Note

For restrictions within a parameterized subroutine block, For restrictions within a parameterized subroutine block, refer to "Restrictions on Formal Parameters Within a Parameterized Subroutine Block" in Section 2 of Chapter 2.
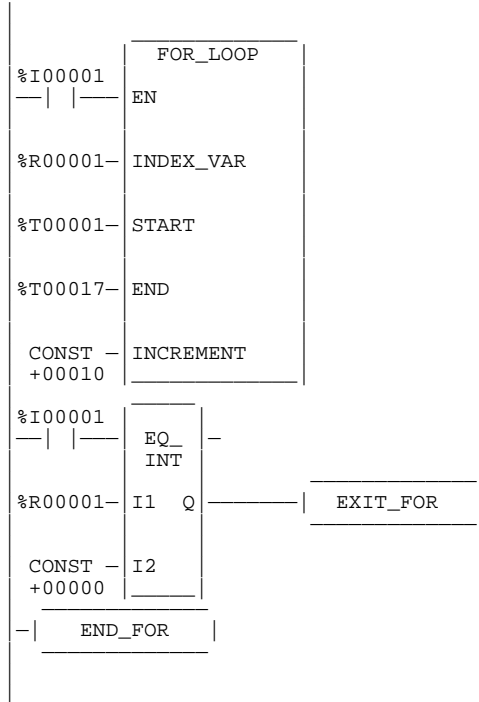
Example:

In the following example, when the enabling input %I00001 is ON, SVCREQ function number 1 is called, with the parameter block located starting at %L00001. Output coil %M00001 is set ON if the operation succeeds.

```
 |                _____
 |%I00001        |      |                                              %M00001
 |—| |——|  SVC_ |——————————————————————————————————————————————————————( )—
 |                | REQ  |
 |                |      |
 | CONST —| FNC  |
 |  00001         |      |
 |                |      |
 |%L00001—| PARM |
 |                |_____|
 |
```
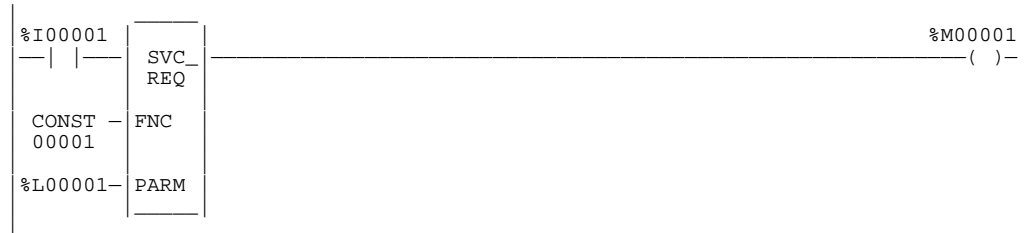
SVCREQ #1: Change/Read Constant Sweep Timer

Use SVCREQ function #1 to:

- Disable **CONSTANT SWEEP** mode.

- Enable **CONSTANT SWEEP** mode and use the old timer value.

- Enable **CONSTANT SWEEP** mode and use a new timer value.

- Set a new timer value only.

- Read **CONSTANT SWEEP** mode state and timer value.

The parameter block has a length of two words.

To disable **CONSTANT SWEEP** mode, enter SVCREQ function #1 with this parameter block:

0	address
ignored	address + 1

To enable **CONSTANT SWEEP** mode, enter SVCREQ function #1 with this parameter block:

1	address
0 or timer value	address + 1

Note

If the timer should use a new value, enter it in the second word. If the timer value should not be changed, enter 0 in the second word. If the timer value does not already exist, entering 0 will cause the function to set the OK output to OFF.

To change the timer value **<u>without</u>** changing the selection for sweep mode state, enter SVCREQ function #1 with this parameter block:

2	address
new timer value	address + 1

To read the current timer state and value without changing either, enter SVCREQ function #1 with this parameter block:

3	address
ignored	address + 1

Note

After using SVCREQ function #1 with the parameter block on the previous page, Release 6 and higher CPUs will provide the return values 0 for Normal Sweep, 1 for Constant Sweep, or 2 for Microcycle. Do not confuse this with the *input* values shown below.

Successful execution will occur, unless:

1. A number other than 0, 1, 2, or 3 is entered as the requested operation:

0	Disable **CONSTANT SWEEP** mode.
1	Enable **CONSTANT SWEEP** mode.
2	Set a new timer value only.
3	Read **CONSTANT SWEEP** mode and timer value. (See Note above).

2. The time value is greater than 2550 ms (2.55 seconds).

3. Constant sweep time is enabled with no timer value programmed, or with an old value of 0 for the timer.

After the function executes, the function returns the timer state and value in the same parameter block references:

0 = disabled	
1 = enabled	address
current timer value	address + 1

If word address + 1 contains the hexadecimal value FFFF, no timer value has ever been programmed.

Example:

This example shows logic in a program block. When enabling 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 PLC. The parameter block is in local memory at location %L00050. Because the MOVE and ADD functions require three horizontal contact positions, the example logic uses discrete internal coil %M00001 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.

```
| OV_SWP   | ____  |                      | ____ |               | ____ |                %M00001
|—| |——    |MOVE_ |—————————————————————  |SVC_ |—————————————————|ADD_ |————————————————( )—
|          |UINT  |                      | REQ |                 | INT |
|          |      |                      |     |                 |     |
| CONST — |IN  Q |—%L00050   CONST — |FNC  |      %L00051—|I1  Q |—%L00051
| 00003    | LEN  |               00001    |     |                 |     |
|          |00001 |                      |     |                 |     |
|          |      |               %L00050—|PARM |       CONST — |I2   |
|          |_____|                      |_____|       00002      |_____|
|
| M00001   | ____  |                      | ____ |
|—| |——    |MOVE_ |—————————————————————  |SVC_ |—
|          |UINT  |                      | REQ |
|          |      |                      |     |
| CONST — |IN  Q |—%L00050   CONST — |FNC  |
| 00001    | LEN  |               00001    |     |
|          |00001 |                      |     |
|          |      |               %L00050—|PARM |
|          |_____|                      |_____|
```
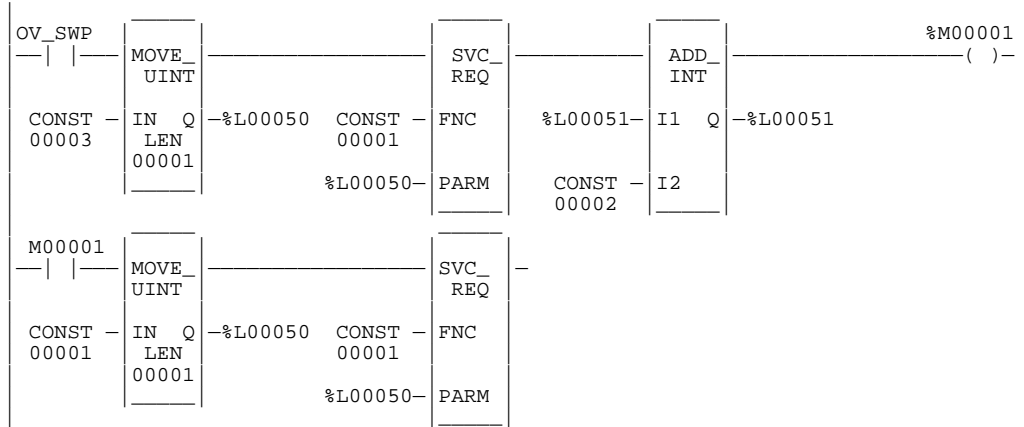
SVCREQ #2: Read Window Values

Use SVCREQ function #2 to obtain the current window mode time values for the programmer communications window, the system communications window, and the background task window. There are three modes for each window:

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 SVCREQ function #3 for the programmer communications window or SVCREQ function #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 PLC 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 PLC is operating in **CONSTANT WINDOW** mode and a particular window's execution time is not defined using the associated SVCREQ 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.

A window is disabled when the time value is zero.

The parameter block has a length of three words:

	High Byte	Low Byte	
Programmer Window	Mode	Value in ms	address
System Communications Window	Mode	Value in ms	address + 1
Background Window			address + 2

All parameters are output parameters. It is not necessary to enter values in the parameter block to program this function. Output values for all three windows are given in milliseconds.

Example:

In the following example, when enabling output %Q00102 is set, the PLC operating system places the current time values of the three windows in the parameter block starting at location %P00010. Additional examples showing the Read Window Values function are included in the next three SYS REQ function descriptions.

```
 %Q00102  |‾‾‾‾‾|
 —| |——   SVC_
           REQ

 CONST —  FNC
 00002
 
 %P00010— PARM
          |_____|
```
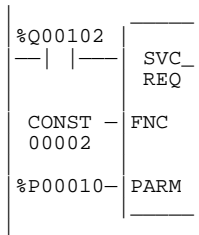
SVCREQ #3: Change Programmer Communications Window Mode and Timer Value

Use SVCREQ function #3 to change the programmer communications window mode and timer value. The change will occur in the CPU sweep following the sweep in which the function is called.

The SVCREQ function #3 will pass power flow to the right unless one of the following occurs:

1. A mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.

2. The PLC is in Microcycle Sweep Mode and a mode other than 1 (Constant) is selected.

The parameter block has a length of one word.

To disable the programmer window, enter SVCREQ function #3 with this parameter block:

High Byte	Low Byte	
0	0	address

To enable the programmer window, enter SVCREQ function #3 with this parameter block:

High Byte	Low Byte	
Mode	Value from 1 to 255 ms	address
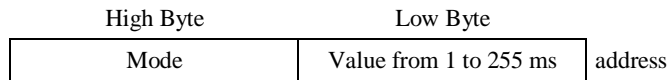
Example:

In the following example, when enabling input %I00125 transitions on the programmer 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.

```
 |      _____
 |%I00125|      |                   +        _____
 |—|↑|——|MOVE_ |——————————+——————|      |—
 |      | UINT |                   | SVC_ |
 |      |      |                   | REQ  |
 |CONST —|IN  Q|—%P00051   CONST —|FNC   |
 |00025  | LEN |            00003  |      |
 |       | 00001|                  %P00051—|PARM  |
 |       |_____|                  |_____|
 |
 |      _____
 |%I00125|      |
 |—|↓|——|MOVE_ |——————————+
 |      | UINT |
 |      |      |
 |CONST —|IN  Q|—%P00051
 |00000  | LEN |
 |       | 00001|
 |       |_____|
 |
```

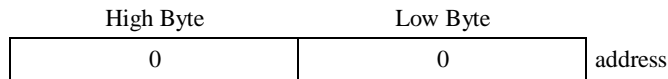## SVCREQ #4: Change System Communications Window Mode and Timer Value

Use SVCREQ function #4 to change the system communications window mode and timer value. The change will occur in the CPU sweep following the sweep in which the function is called.

The SVCREQ function #4 will pass power flow to the right unless one of the following occurs:

1. A mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.

2. The PLC is in Microcycle Sweep Mode and a mode other than 1 (Constant) is selected.

The parameter block has a length of one word.

To disable the system communications window, enter SVCREQ function #4 with this parameter block:
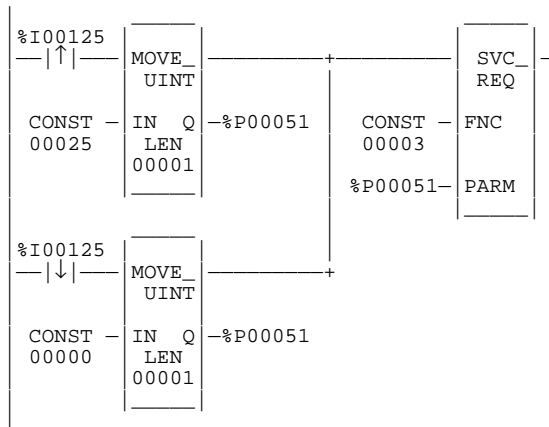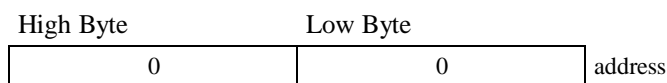
High Byte	Low Byte	
0	0	address

To enable the system communications window, enter SVCREQ function #4 with this parameter block:

High Byte	Low Byte	
Mode	Value from 1 to 255 ms	address
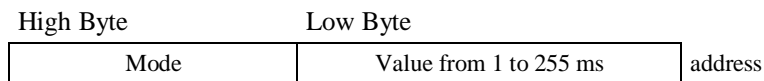
Example:

In the following example, when enabling output %Q00125 transitions on the mode and timer value of the system 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 %P00051. Since the mode and timer for the system communications window is the second value in the parameter block returned from the Read Window Values function (function #2), the location of the existing window time for the system communications window is in the low byte of %P00052.

```
 %Q00125     _____            _____                   _____
 —|↑|——     | SVC_ |          | AND_ |                 | AND_ |
            |  REQ |          | WORD |                 | WORD |
            |      |          |      |                 |      |
 CONST —   | FNC  | %P00052—| I1  Q|—%P00060  %P00052—| I1  Q|—%P00061
 00002     |      |          |      |                 |      |
           |      |          |      |       CONST —   | I2   |
 %P00051— | PARM |   CONST —| I2   |       FF00       |      |
           |_____|   00FF  |_____|                 |_____|

 %Q00125     _____            _____                   _____
 —|↑|——     | LT  |   +————————————     | OR  |         | SVC_ | —
            | UNIT|   |                 | WORD|         | REQ_ |
            |     |   |                 |     |         |      |
 %P00060—  | I1  Q|————————+    %P00061—| I1  Q|—%P00052 CONST —| FNC  |
            |     |                     |     |         00004   |      |
 CONST —   | I2  |          CONST —    | I2  |                 |      |
 00025     |_____|          00025      |_____|         %P00052—| PARM |
                                                               |_____|
```

SVCREQ #5: Change Background Task Window Mode and Timer Value

Use the SVCREQ function # 5 to change the background task window mode and timer value. The change will occur during the same CPU sweep in which the function is called.

Use SVCREQ function #4 to enable or disable the system communications window. The change will occur in the same CPU sweep in which the function is called.

The SVCREQ function #4 will pass power flow to the right unless one of the following occurs:

1. A mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.

2. The PLC is in Microcycle Sweep Mode and a mode other than 1 (Constant) is selected.

SVCREQ function #5 always passes power flow to the right. The parameter block has a length of 1 word.

To disable the background task window, enter SVCREQ function 5 with this parameter block:

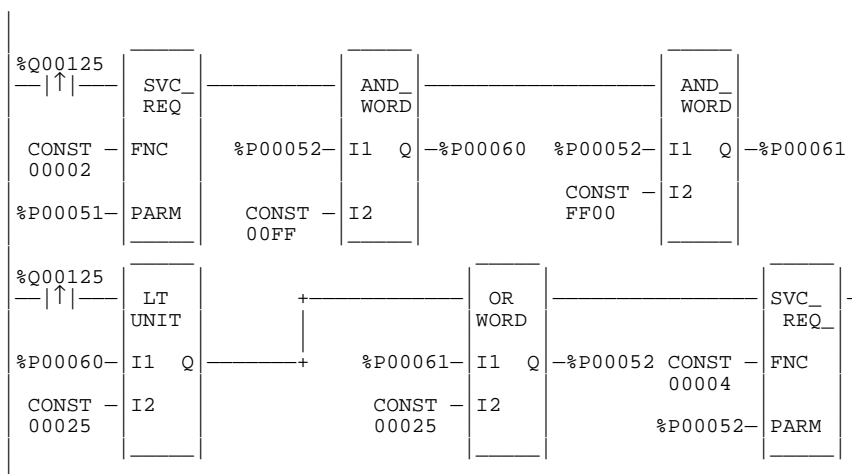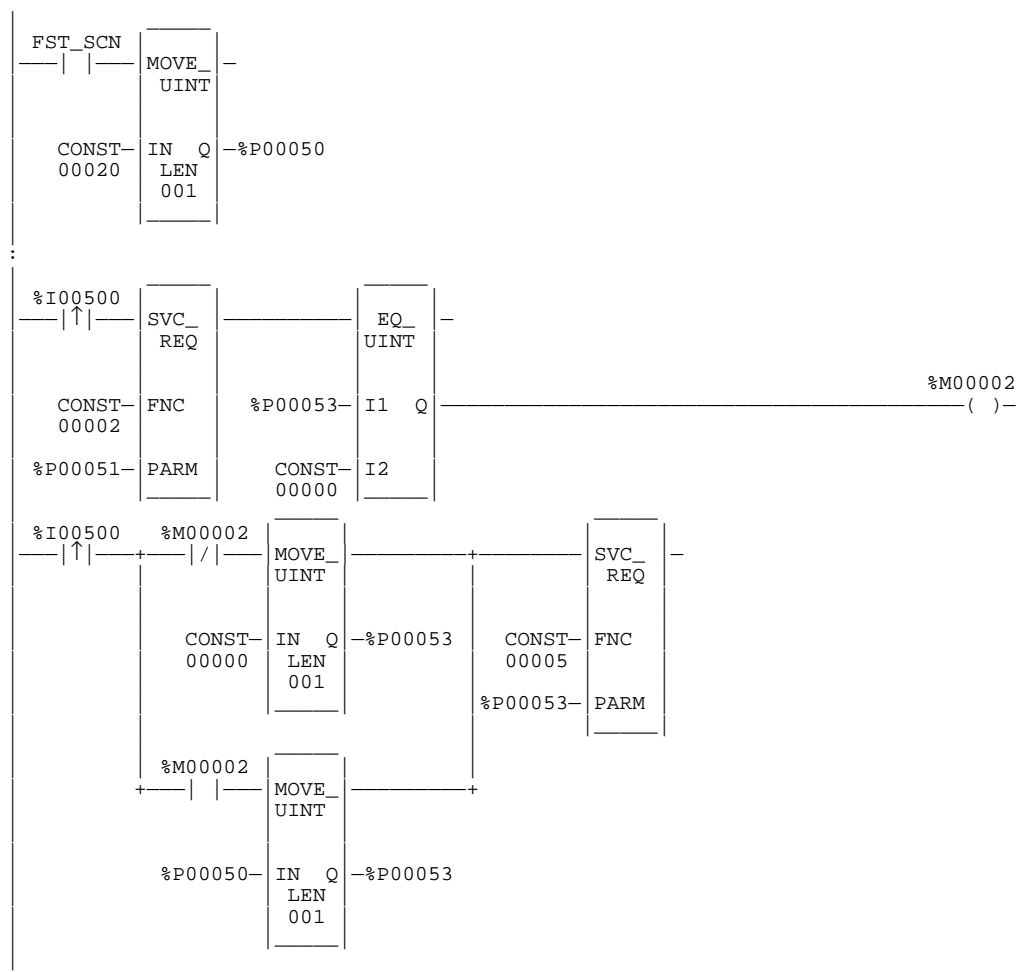| High Byte	Low Byte	
0	0	address

To enable the background task window, enter SVCREQ function #5 with this parameter block:

High Byte	Low Byte	
Mode	Value from 1 to 255 ms	address

Example:

In the following example, when enabling contact FST_SCN is set in the first scan, the MOVE function establishes a default value of 20 ms 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 system communications window is %P00053.

```
    FST_SCN     _____
  ───| |─────| MOVE_ |─
              |  UINT |
              |       |
     CONST──| IN   Q |─%P00050
      00020 | LEN    |
            | 001    |
            |_____|

:

   %I00500    _____                   _____
  ───|↑|────| SVC_ |───────────────| EQ_  |─
            |  REQ |                |  UINT |
            |      |                |       |                                         %M00002
     CONST──| FNC  |     %P00053──| I1   Q |─────────────────────────────────────────( )─
      00002 |      |              |         |
   %P00051──| PARM |     CONST──| I2      |
            |_____|      00000 |_____|

   %I00500   %M00002   _____                  _____
  ───|↑|────+───| / |─| MOVE_ |──────────+──| SVC_ |─
            |           |  UINT |          |     |  REQ |
            |           |       |          |     |      |
            |  CONST──| IN   Q |─%P00053  CONST──| FNC  |
            |   00000 | LEN    |           00005 |      |
            |         | 001    |        %P00053──| PARM |
            |         |_____|                 |_____|
            |
            |  %M00002   _____
            +───| |────| MOVE_ |──────────+
            |           |  UINT |          |
            |           |       |
            |%P00050──| IN   Q |─%P00053
            |           | LEN    |
            |           | 001    |
            |           |_____|
            |
```

SVCREQ #6: Change/Read Checksum Task State and Number of Words to Checksum

Use SVCREQ function #6 to read the current word count or set a new word count. By default, 16 words will be checked. Successful execution will occur, unless some number other than 0 or 1 is entered as the requested operation (see below).

The parameter block has a length of two words.

To read the current word count, enter SVCREQ function #6 with this parameter block:

0	address
ignored	address + 1

After the function executes, the function returns the current checksum 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.

0	address
current word count	address + 1

To set a new word count, enter SVCREQ function #6 with this parameter block:

1	address
new word count	address + 1

Entering 1 causes the PLC to adjust the number of words to be checksummed to the value given in the second word of the parameter block, rounded up to a multiple of 8. To disable checksumming, set the count to zero.

Example:

In the following 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 PLC 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 PLC to set the new word count.

```
|
|  FST_SCN      _____                                   _____
|——| |——————|  XOR_ |——————————————————————————|  MOVE_ |
|              | WORD |                          |  UINT  |
|              |      |                          |        |
| %L00150 —————|I1  Q|—%L00150     CONST —|IN  Q|—%L00152
|              |      |             00001  |LEN  |
|              |      |                    |00001|
| %L00150 —————|I2   |                     |_____|
|              |_____|
|
.
.
|
|  %I00137      _____                _____                           _____
|——| ↑ |———————|  SVC_ |————————————|  ADD_ |———————————————————————|  SVC_ |—
|              |  REQ  |            |  UINT  |                       |  REQ  |
|              |       |            |        |                       |       |
|     CONST ———|FNC    | %L00151 —|I1  Q|—%L00153   CONST —|FNC    |
|     00006    |       |          |        |           00006  |       |
|              |       |          |        |                 |       |
|   %L00150 ———|PARM   |  CONST —|I2   |             %L00152 —|PARM   |
|              |_____| |  00016    |_____|                     |_____|
|
```
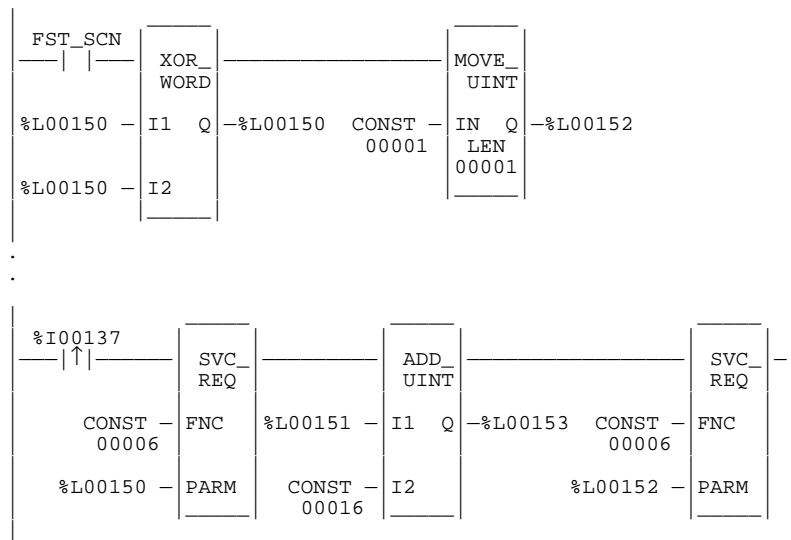
The example parameter blocks are located at address %L00150. They have the following content:

0 = read current count	%L00150
hold current count	%L00151

1 = set current count	%L00152
hold new count for set	%L00153

SVCREQ #7: Change/Read Time-of-Day Clock State and Values

Use SVCREQ function #7 to read or set the time-of-day clock in the PLC.

The length of the parameter block depends on the data format. Numeric and unpacked BCD each require nine words. BCD format requires six words. Packed ASCII requires twelve words.

0 = read time and date	address (word 1)
1 = set time and date	
0 = numeric data format	address + 1 (word 2)
1 = BCD format	
2 = unpacked BCD format	
3 = packed ASCII format	
data	address + 2 to end (word 3)

In word 1, specify whether the function should read or change the values.

> 0   =   read
>
> 1   =   change (write)

In word 2, specify a data format (i.e., the format used to read or write):

> 0   =   numeric
>
> 1   =   BCD
>
> 2   =   unpacked BCD
>
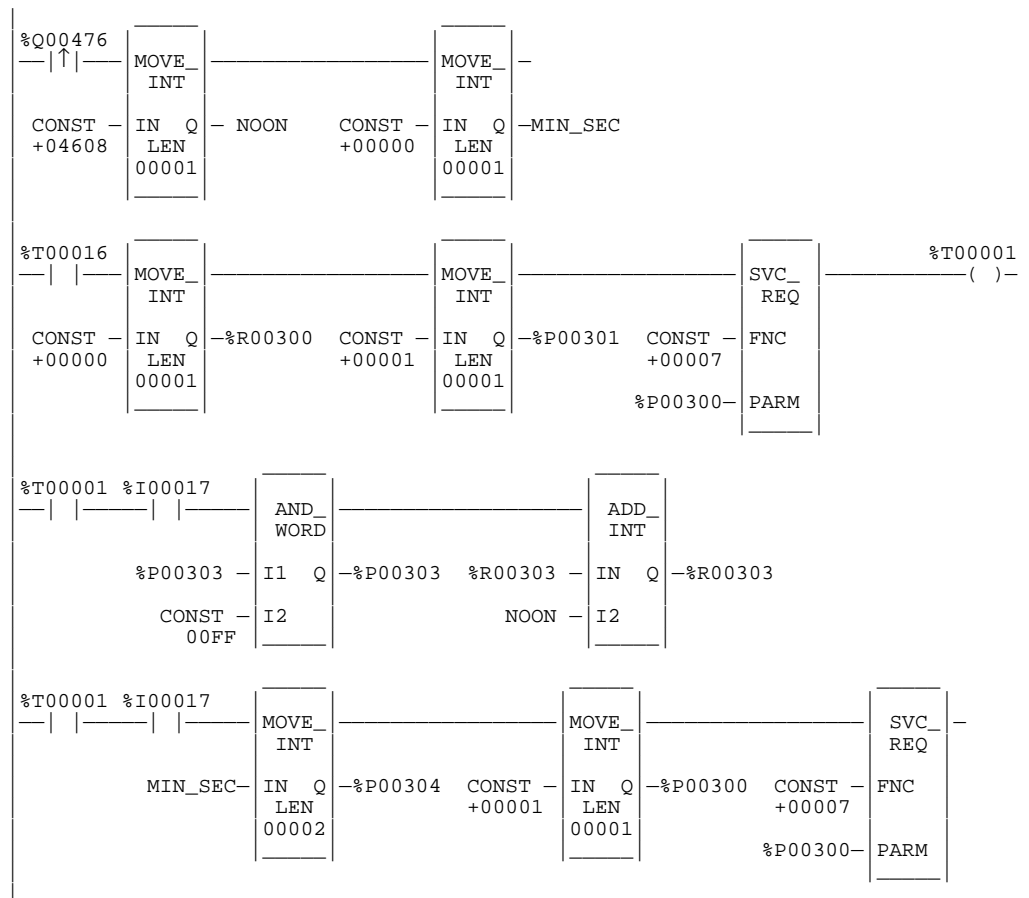> 3   =   packed ASCII with embedded spaces and colons

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) through (address + 8) of the parameter block are ignored on input.

Successful execution will occur unless:

- Some number other than 0 or 1 is entered as the requested operation (see below).

- An invalid data format is specified.

- The data provided is not in the expected format.

Example:

In the following example, when output %Q00476 is on, a parameter block for the time-of-day clock is built to first request the current date and time, and then set the clock to 12 noon using the BCD format. The parameter block is located at global data location %P00300. Array NOON has been set up elsewhere in the program to contain the values 12, 0, and 0. (Array NOON must also contain the data at %R0300.) The BCD format requires six contiguous memory locations for the parameter block.

```
  |        _____                              _____
  |%Q00476|MOVE_|                            |MOVE_|-
  |—|↑|——|INT  |————————————————————|INT  |
  |        |     |                            |     |
  |CONST —|IN  Q|— NOON       CONST —|IN  Q|—MIN_SEC
  | +04608 |LEN  |                +00000 |LEN  |
  |        |00001|                            |00001|
  |        |_____|                            |_____|
  |
  |        _____                       _____                                   _____
  |%T00016|MOVE_|                     |MOVE_|                                 |SVC_ |          %T00001
  |—| |——|INT  |————————————|INT  |————————————————|REQ  |—————————( )—
  |        |     |                     |     |                                 |     |
  |CONST —|IN  Q|—%R00300 CONST —|IN  Q|—%P00301  CONST —|FNC  |
  | +00000 |LEN  |                +00001 |LEN  |              +00007 |     |
  |        |00001|                     |00001|                                 |     |
  |        |_____|                     |_____|                    %P00300—|PARM |
  |                                                                              |_____|
  |
  |                  _____                              _____
  |%T00001 %I00017 |AND_ |                            |ADD_ |
  |—| |———| |——|WORD |————————————————————|INT  |
  |                  |     |                            |     |
  |        %P00303 —|I1  Q|—%P00303  %R00303 —|IN  Q|—%R00303
  |                  |     |                            |     |
  |        CONST —|I2   |                  NOON —|I2   |
  |          00FF  |_____|                            |_____|
  |
  |                  _____                              _____                          _____
  |%T00001 %I00017 |MOVE_|                            |MOVE_|                         |SVC_ |-
  |—| |———| |——|INT  |————————————————————|INT  |————————————|REQ  |
  |                  |     |                            |     |                          |     |
  |        MIN_SEC—|IN  Q|—%P00304  CONST —|IN  Q|—%P00300  CONST —|FNC  |
  |                  |LEN  |                +00001 |LEN  |              +00007 |     |
  |                  |00002|                            |00001|                          |     |
  |                  |_____|                            |_____|            %P00300—|PARM |
  |                                                                                   |_____|
```

Parameter Block Contents

Parameter block contents for the four different data formats are shown on the following pages. For all data formats, hours are stored in a 24-hour format, and the day of the week is a numeric value.

Value	Day of the Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

To Change/Read Date and Time Using Numeric Format:

In numeric format, 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 numeric format, enter SVCREQ function #7 with this parameter block:

Example output parameter block:
Read Date and Time in Numeric Format
(Weds, June 15, 1988 at 12:15:30)

High Byte	Low Byte		
1 = change or	0 = read	address	0
0		address + 1	0
year		address + 2	88
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

To Change/Read Date and Time Using BCD Format:

In BCD format, each of the time and date items occupies a single byte. This format requires 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).

Example output parameter block:
Read Date and Time in BCD Format
(Mon, July 3, 1988 at 2:45:30 p.m.)

High Byte	Low Byte			
1 = change or	0 = read	address		0
1		address + 1		1
month	year	address + 2	07	88
hours	day of mo.	address + 3	14	03
seconds	minutes	address + 4	30	45
(null)	day of week	address + 5	00	02

To Change/Read Date and Time Using Unpacked BCD Format:

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.

Example output parameter block:
Read Date and Time in Unpacked BCD Format
(Thurs, Dec. 28, 1989 at 9:34:57)

High Byte	Low Byte	
1 = change or	0 = read	address
2		address + 1
	year	address + 2
	month	address + 3
	day of month	address + 4
	hours	address + 5
	minutes	address + 6
	seconds	address + 7
	day of week	address + 8

0h	
2h	
08h	09h
01h	02h
02h	08h
00h	09h
03h	04h
05h	07h
00h	05h

To Change/Read Date and Time Using Packed ASCII with Embedded Colons Format:

In Packed ASCII format, each digit of the time and date items is an ASCII formatted byte. In addition, spaces and colons are embedded into the data to permit it to be transferred unchanged to a printing or display device. This format requires 12 words.

Example output parameter block:
Read Date and Time in Packed ASCII Format
(Tues, Oct. 2, 1989 at 23:13:00)

High Byte	Low Byte	
1 = change or	0 = read	address
3		address + 1
year	year	address + 2
month	(space)	address + 3
(space)	month	address + 4
day of mo.	day of mo.	address + 5
hours	(space)	address + 6
:	hours	address + 7
minutes	minutes	address + 8
seconds	:	address + 9
(space)	seconds	address + 10
day of week	day of week	address + 11

0h	
3h	
39h	38h
31h	20h
20h	30h
32h	30h
32h	20h
3Ah	33h
33h	31h
30h	3Ah
20h	30h
33h	30h

SVCREQ #8: Reset Watchdog Timer

Use SVCREQ function #8 to reset the watchdog timer during the sweep. When the watchdog timer expires, the PLC shuts down without warning. This function allows the timer to keep going during a time-consuming task (for example, while waiting for a response from a communications line).

<div style="text-align:center">

Caution

</div>

Be sure that restarting the watchdog timer does not adversely affect the controlled process.

This function has no associated parameter block; however, the programming software requires that an entry be made for PARM. Enter any appropriate reference here; it will not be used.

Example:

In the following example, when enabling output %Q00127 or input %I01476 or internal coil %M00010 is set, the watchdog timer is reset.

```
 |                             _____
 |%Q00127                     |     |
 |—| |——+——————————|  SVC_| —
 |                     |  REQ |
 |%I01476              |     |
 |—| |——      CONST —| FNC  |
 |            00008    |     |
 |%M00010              |     |
 |—| |——+ %AI0001—| PARM |
 |                     |_____|
 |
```
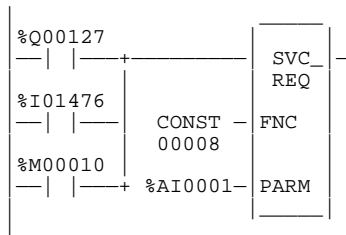
SVCREQ #9: Read Sweep Time from Beginning of Sweep

Use SVCREQ function #9 to read the time in milliseconds since the start of the sweep. The data format is in unsigned 16-bit integer.

The parameter block is an output parameter block only; it has a length of one word.

| time since start of sweep | address |

Example:

In the following example, the elapsed time from the start of the sweep is always read into location %P00200. If it is greater than the value in %P00201, internal coil %M00200 is turned on.

```
|                 _____                  _____
|                |     |                |     |  _
|_____| SVC_|_____| GT_ |
|                | REQ |                | UINT|
|                |     |                |     |            %M00200
|   CONST ──| FNC|      %P00200──| I1  Q |────────────────( )─
|   00009   |     |               |     |
|           |     |               |     |
|  %P00200──| PARM|      %P00201──| I2  |
|           |_____|               |_____|
|
```
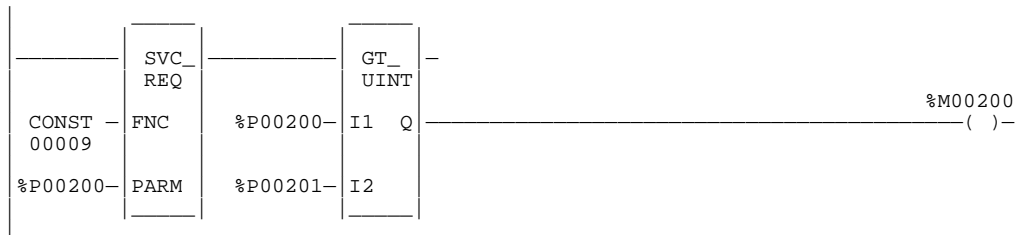
Note

This SVCREQ has a different meaning with Microcycle mode. It reflects the time from the beginning of the sweep in which the program was scheduled to begin execution.

SVCREQ #10: Read Folder Name

Use SVCREQ function #10 to read the name of the currently-executing folder.

The output parameter block has a length of four words. It returns eight ASCII characters; the last is a null character (00h). If the program name has fewer than seven characters, null characters are appended to the end.

Low Byte	High Byte	
character 1	character 2	address
character 3	character 4	address + 1
character 5	character 6	address + 2
character 7	00	address + 3

Example:

In the following example, when enabling input %I00301 transitions off, register location %R00099 is loaded with the value 10, which is the function code for the Read Folder Name function. The Program Block READ_ID is then called to actually retrieve the folder name. The parameter block is located at address %R00100. READ_ID is also used in the next example.

```
     |        _____                    _____
%I00301       |       |                  |         |
—|↓|——| MOVE_ |——————————| READ_ID |
     |        |  UINT |                  |_____|
     |        |       |
 CONST —| IN   Q |—%R00099
 00010   |  LEN   |
     |        | 00001 |
     |        |_____|
     |
.
.
.
Program Block READ_ID

     |        _____
     |        |       |
————| SVC_  |—
     |        |  REQ  |
     |        |       |
%R00099—| FNC  |
     |        |       |
%R00100—| PARM |
     |        |_____|
     |
```
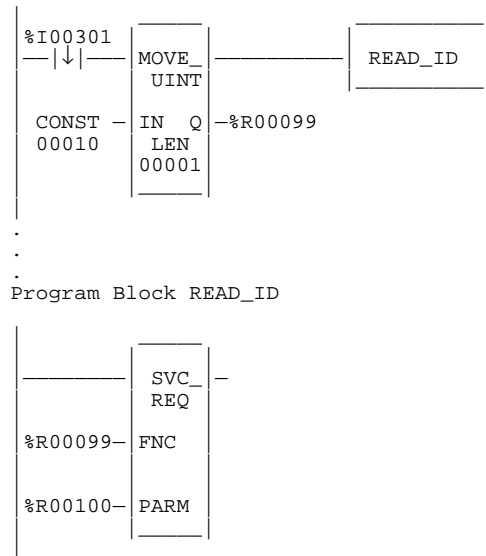
SVCREQ #11: Read PLC ID

Use SVCREQ function #11 to read the name of the Series 90 PLC executing the program.

The output parameter block has a length of four words. It returns eight ASCII characters; the last is a null character (00h). If the PLC ID has fewer than seven characters, null characters are appended to the end.

Low Byte	High Byte	
character 1	character 2	address
character 3	character 4	address + 1
character 5	character 6	address + 2
character 7	00	address + 3

Example:

In the following example, when enabling input %I00302 transitions off, register location %R00099 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 actually retrieve the ID. The parameter block is located at address %R00100. Except for the enabling contact and function number, this is the same code used in the previous example.

```
|          _____                _____
|%I00303  |      |              |         |
|——|↓|————|MOVE_ |——————————————| READ_ID |—
|         | UINT |              |_____|
|         |      |
| CONST —|IN   Q|—%R00099
| 00011  | LEN  |
|        | 00001|
|        |_____|
|
.
.
.
Program Block READ_ID
|          _____
|         |      |
|—————————|SVC_  |—
|         | REQ  |
|%R00099—|FNC   |
|         |      |
|%R00100—|PARM  |
|        |_____|
|
```
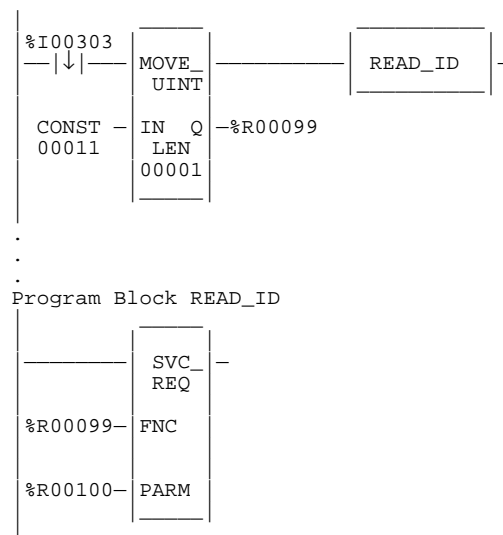
SVCREQ #12: Read PLC Run State

Use SVCREQ function #12 to read the current RUN state of the PLC CPU.

The parameter block is an output parameter block only; it has a length of one word.

1 = run/disabled
2 = run/enabled

address

Example:

In the following example, the PLC run state is always read into location %L01002. If the state is Run/Disabled, the CALL function calls program block DISPLAY.

```
 |          _____          _____
 |         | SVC_ |        | EQ_  | _
 |―――――――――| REQ  |――――――――| UINT |
 |         |      |        |      |          _____
 | CONST ―| FNC  | DISABL―| I1  Q|―――――――――| DISPLAY   |_
 | 00012  |      |        |      |          |_____|
 |         |      |        |      |
 | %L01002―| PARM | %L01002―| I2   |
 |         |_____|        |_____|
 |
```
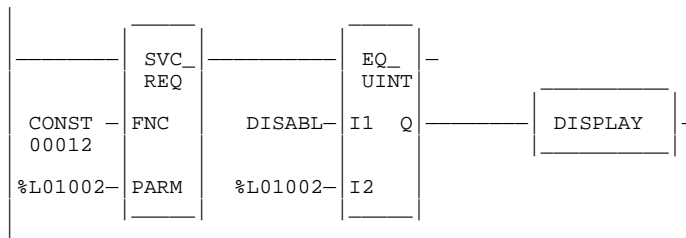
SVCREQ #13: Shut Down (Stop) PLC

Use SVCREQ function #13 to stop the PLC at the end of the current sweep. All outputs will go to their designated default states at the beginning of the next PLC sweep. An informational fault is placed in the PLC fault table, noting that a "SHUT DOWN PLC" function block was executed.

This function has no parameter block; however, Logicmaster 90-70 software requires an entry be made for PARM.

Example:

In the following example, when enabling input %I00001 is set and a lost rack fault occurs, SVCREQ function #13 executes. Since no parameter block is needed, the PARM input is not used; however, the programming software requires that an entry be made for PARM.

This example uses a JUMP to the end of the program to force a shutdown if the Shutdown PLC function executes successfully. This JUMP and LABEL are needed because the transition to **STOP** mode does not occur until the end of the sweep in which the function executes.

```
|
|%I00001 LOS_RCK                          _____
|—| |———|↑|————————————————| SVC_ |————————————————————>> END_PRG
|                               | REQ  |
|                               |      |
|                    CONST —| FNC  |
|                      00013 |      |
|                               |      |
|                  %R00100 —| PARM |
|                               |_____|
|
.
.
.
|
| END_PRG
|
| [END OF PROGRAM]
|
```
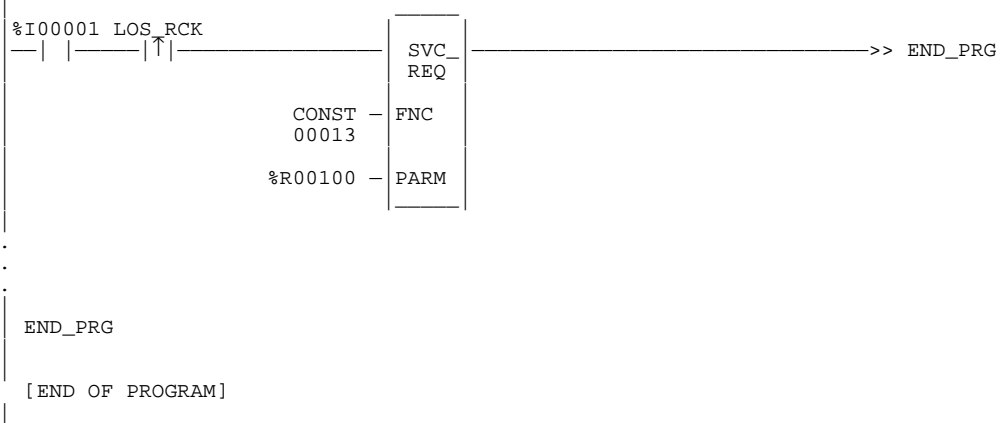
SVCREQ #14: Clear Fault Tables

Use SVCREQ function #14 to clear either the PLC or I/O fault table. The SVCREQ output is set
ON, unless some number other than 0 or 1 is entered as the requested operation (see below).

The parameter block is an input parameter block only; it has a length of one word.

0 = Clear PLC fault table.	address
1 = Clear I/O fault table.	

Example:

In the following example, when input %I00346 is on and input %I00349 transitions on, the PLC
fault table is cleared. When input %I00347 is on and input %I00349 transitions on, the I/O fault
table is cleared. When input %I00348 is on and input %I00349 transitions on, both are cleared.

The parameter block for the PLC fault table is located at %P00500; for the I/O fault table, the
parameter block is located at %P00550. Both parameter blocks are set up elsewhere in the
program.

```
|        _____                              _____
%I00349 |      |                            |      |
—|↑|———| MOVE_|————————————————————| MOVE_|—
|       |  INT |                     |  INT |
|       |      |                     |      |
CONST —|IN   Q|—%P00500   CONST —|IN   Q|—%P00550
+00000 | LEN  |           +00001 | LEN  |
       |00001 |                  |00001 |
       |_____|                  |_____|


|                                  _____
%I00349 %I00346                   |      |
—|↑|———+——| |——+————————————| SVC_ |—
|      |         |           | REQ  |
|      %I00348|              |      |
|      —| |——+   CONST —|FNC   |
|              00014       |      |
|                         |      |
|             %P00500—|PARM  |
|      %I00347            |_____|
|      —| |——+
|      %I00348|            _____
|      +——| |——+—————————| SVC_ |—
|                         | REQ  |
|                         |      |
|             CONST —|FNC   |
|             00014       |      |
|                         |      |
|             %P00550—|PARM  |
|                         |_____|
```
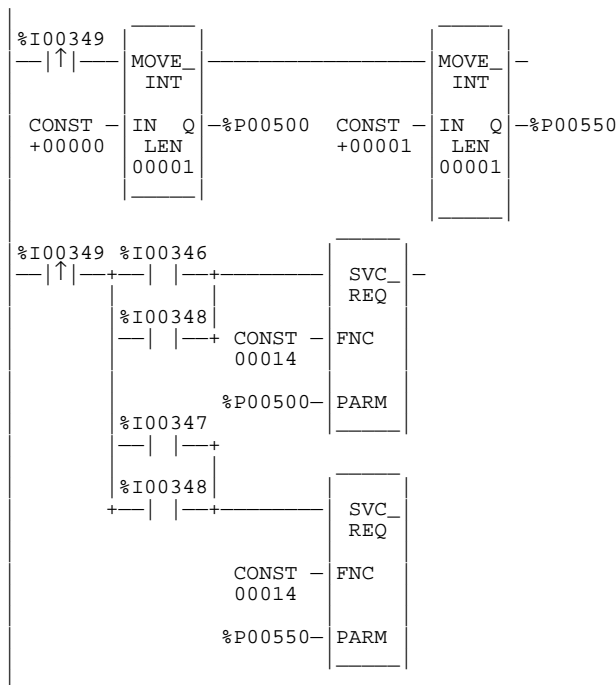
SVCREQ #15: Read Last-Logged Fault Table Entry

Use SVCREQ function #15 to read the last entry logged in either the PLC or I/O fault table. The SVCREQ output is set ON, unless some number other than 0, 1, 80, or 81 (hexadecimal base) is entered as the requested operation (see below) or the fault table is empty. (For more information on fault table entries, refer to chapter 3, "Fault Explanation and Correction.")

The parameter block has a length of 22 words. The input parameter block has this format:

0 = Read PLC fault table.	address
1 = Read I/O fault table.	
80h = Read extended PLC fault table.	
81h = Read extended I/O fault table.	

The format for the output parameter block depends on whether the function reads data from the PLC fault table, the I/O fault table, the extended PLC fault table (see next page), or the extended I/O fault table (see next page).

PLC Fault Table Output Format

High Byte	Low Byte	
0		address
	long/short	address + 1
spare		address + 2
PLC fault address		address + 3
		address + 4
fault group and action		address + 5
error code		address + 6
		address + 7
		address + 8
		address + 9
		address + 10
		address + 11
fault specific data		address + 12
		address + 13
		address + 14
		address + 15
		address + 16
		address + 17
		address + 18
		address + 19
time stamp		address + 20
		address + 21

I/O Fault Table Output Format

High Byte	Low Byte
1	
	long/short
reference	
I/O fault address	
fault group and action	
fault type	fault category
	fault description
fault specific data	
time stamp	

In the least significant byte of word address + 1, the Long/Short indicator defines the quantity of fault specific data present in the fault entry.

As mentioned on the previous page, the format for the output parameter block depends on whether the function reads data from the PLC fault table, the I/O fault table, the extended PLC fault table, or the extended I/O fault table. The extended versions are shown below:

Extended PLC Fault Table Output Format

High Byte	Low Byte	
80h		address
	long/short	address + 1
spare		address + 2
PLC fault address		address + 3
		address + 4
fault group and action		address + 5
error code		address + 6
		address + 7
		address + 8
		address + 9
		address + 10
		address + 11
fault specific data		address + 12
		address + 13
		address + 14
		address + 15
		address + 16
		address + 17
		address + 18
		address + 19
time stamp		address + 20
		address + 21
		address + 22
reserved		address + 23

Extended I/O Fault Table Output Format

High Byte	Low Byte
81h	
	long/short
reference	
I/O fault address	
fault group and action	
fault type	fault category
	fault description
fault specific data	
time stamp	
reserved	

Since the extended fault format is 2 words (4 bytes) longer than the standard format, the amount of reference memory needed to read the last fault has increased. The following table shows the number of bytes needed to read the last fault entry in each of the fault formats.

Fault Table	Number of Bytes
PLC fault table Extended PLC fault table	00 = 8 bytes (short 01 = 24 bytes (long)
I/O fault table Extended I/O fault table	02 = 5 bytes (short 03 = 21 bytes (long)

Example 1:

In the following example, when input %I00251 is on and input %I00250 transitions on, the last entry in the PLC fault table is read into the parameter block. When input %I00251 is off and input %I00250 transitions on, the last entry in the I/O fault table is read into the parameter block. The parameter block is located in global memory at location %P00600.

```
  %I00250    %I00251   _____
 ―|↑|――――+――| |――|MOVE_ |――――――――――――+
  |       |        | UINT |                         |
  |       |        |      |                         |
  |       |  CONST ―|IN  Q|―%P00600                 |
  |       |  00000  | LEN |                         |
  |       |        |00001 |                         |
  |       |        |_____|                         |
  |                                        _____   |
  |                                       |SVC_ |―   |
  |  %I00251   _____                     | REQ |    |
  +――| / |――|MOVE_ |――――――――――――+  CONST ―|FNC  |    |
  |        | UINT |                00015 |     |    |
  |        |      |                      |     |    |
  |  CONST ―|IN  Q|―%P00600       %P00600―|PARM |    |
  |  00001  | LEN |                       |_____|    |
  |        |00001 |                                  |
  |        |_____|                                  |
  |                                                  |
```
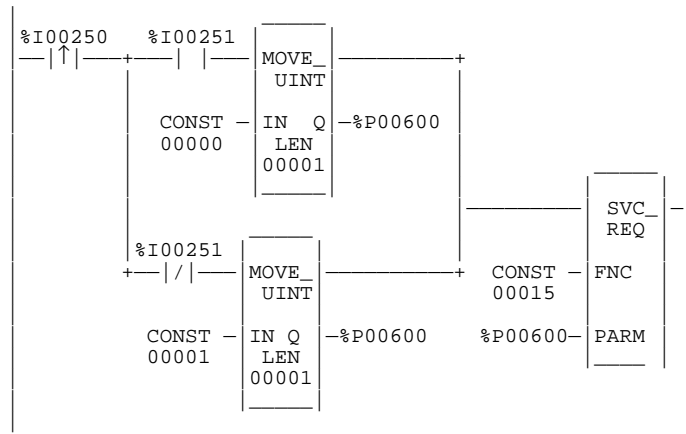
Example 2:

In the next example, the PLC is shut down when any fault occurs on an I/O module except when the fault occurs on boards 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 sweep. The contact IO_PRES, when set, indicates that the I/O fault table contains an entry. The PLC CPU sets the up transition contact in the sweep after the fault logic places a fault in the table. If faults are placed in the table in two consecutive sweeps, the up transition contact is set for two consecutive sweeps.

The example uses a parameter block located at global memory %P00600. After the SVCREQ function executes, the fourth, fifth, and sixth words of the parameter block contain the address of the I/O module that faulted:

1		%P00600
long/short		%P00601
reference address		%P00602
rack number	slot number	%P00603
I/O bus no.	bus address	%P00604
point address		%P00605

fault data

In the program, the EQ_UINT blocks compare the rack/slot address in the table to hexadecimal constants. The internal coil %M00007 is turned on when the rack/slot where the fault occurred meets the criteria specified above. If %M00007 is on, its normally closed contact is off, preventing the shutdown. Conversely, if %M00007 is off because the fault occurred on a different module, the normally closed contact is on and the shutdown occurs.

```
            _____
FST_SCN    |      |
—| |———| MOVE_ |—
           | UINT |
           |      |
 CONST —|IN  Q|—%P00600
 00001     | LEN  |
           |00001 |
           |_____|
  .
  .
            _____                  _____
IO_PRES    |      |                |      |
—|↑|———| SVC_  |————+————————| EQ_  |—
           | REQ  |                | UINT |                            %M00007
           |      |                |      |                        ——————( )—
 CONST —|FNC  |       %P00603—|I1  Q|———————+
 00015     |      |                |      |               |
           |      |        CONST —|I2   |               |
%P00600—|PARM |        0109    |_____|               |
           |_____|                                      |
                                  _____                 |
                             +————| EQ_  |—              |
                             |    | UINT |               |
                             |    |      |               |
                     %P00603—|I1  Q|————————+
                             |    |      |
                      CONST—|I2   |
                      0009  |_____|
            _____
IO_PRES %M00007|      |
—|↑|———| / |—| SVC_ |—
                  | REQ  |
                  |      |
        CONST —|FNC  |
        00013     |      |
                  |      |
               —|PARM |
                  |_____|
```
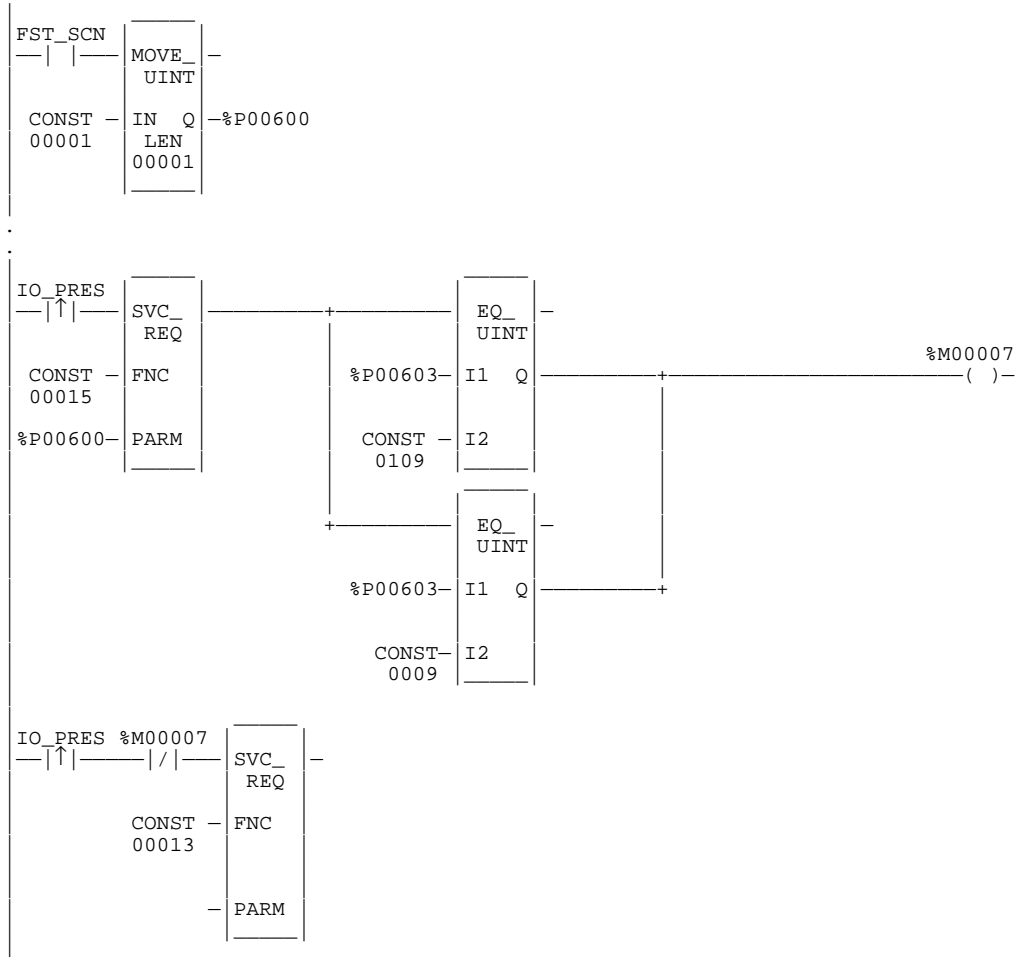
SVCREQ #16: Read Elapsed Time Clock

Use SVCREQ function #16 to read the value of the system's elapsed time clock. This clock tracks elapsed time in seconds since the PLC powered on. The timer will roll over approximately once every 100 years.

The parameter block is an output parameter block only; it has a length of three words.

seconds from power on (low order)	address
seconds from power on (high order)	address + 1
100 microsecond ticks	address + 2

The first two words are the elapsed time in seconds. The last word is the number of 100 microsecond ticks in the current second.

Example:

In the following example, when internal coil %M00233 transitions on, the value of the elapsed time clock is read and coil %M00234 is set. When it transitions off, the value is read again. After the off transition, the difference between the values is calculated and the result is stored in register memory at location %R00250.

The parameter block for the first read is at %P00127; for the second read, at %P00131. The calculation ignores the number of hundred microsecond ticks and the fact that the DINT type is actually a signed value. The calculation is correct until the time since power on reaches approximately 50 years.

```
 %M00233                    ┌─────┐                                                    %M00234
───┤↑├──────┤           │ SVC_│────────────────────────────────────────────────────────(S)──
                        │ REQ │
                        │     │
     CONST ─┤FNC │
       00016 │     │
                        │     │
     %P00127─┤PARM│
                        └─────┘


 %M00233                    ┌─────┐              %M00234        ┌─────┐                  %M00234
───┤↓├──────┤           │ SVC_│──────────────────┤ ├────────┤ SUB_│────────────────────(R)──
                        │ REQ │                              │ DINT│
                        │     │                              │     │
     CONST ─┤FNC │                  %P00131─┤I1  Q├─%R00249
       00016 │     │                              │     │
                        │     │                  %P00127─┤I2   │
     %P00131─┤PARM│                              │     │
                        └─────┘                              └─────┘
```
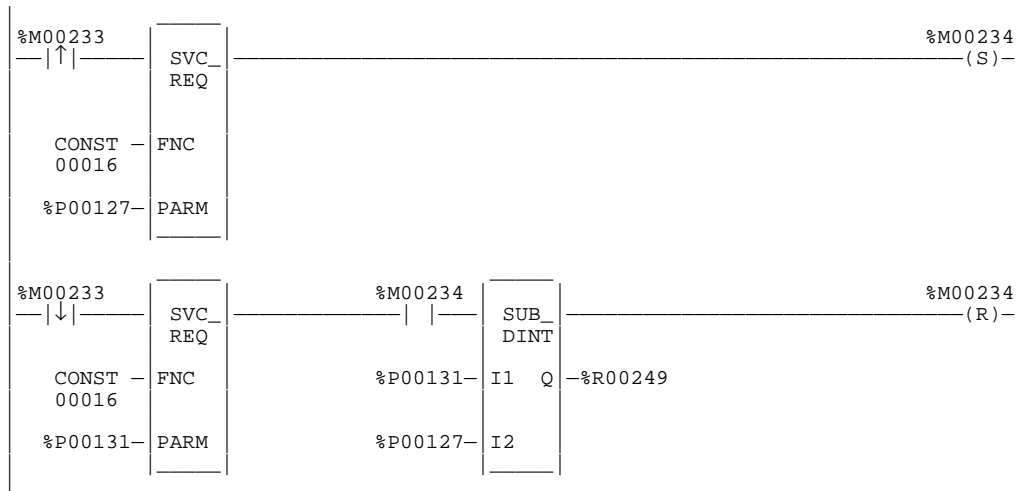
SVCREQ #17: Mask/Unmask I/O Interrupt

Use SVCREQ function #17 to mask or unmask an interrupt from an input board. When an interrupt is masked, the PLC CPU will 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.

0 = unmask input 1 = mask input	address
memory type	address + 1
reference (offset)	address + 2

"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
```

Successful execution will occur unless:

- Some number other than 0 or 1 is entered as the requested operation.

- The memory type of the input to be masked or unmasked is not %I or %AI memory.

- The I/O board is not an appropriate Series 90 input 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.

Example 1:

In the following example, when %I00346 transitions on, interrupts from input %I00033 are masked. The parameter block at %P00347 is set up on the first sweep.

```
|         _____                          _____
FST_SCN  |      |                        |      |
—| |——— | MOVE_ |—————————————————————— | MOVE_ |—
         | UINT |                        | UINT |
         |      |                        |      |
CONST — | IN  Q |—%P00348   CONST — | IN  Q |—%P00349
00070    | LEN  |            00033    | LEN  |
         | 00001|                     | 00001|
         |_____|                     |_____|

|         _____                          _____
%I00346  |      |                        |      |
—|↑|——— | MOVE_ |—————————————————————— | SVC_ |—
         | UINT |                        | REQ  |
         |      |
CONST — | IN  Q |—%P00347   CONST — | FNC
00001    | LEN  |            00017
         | 00001|
         |_____|                     %P00347— | PARM
                                                |_____|
```
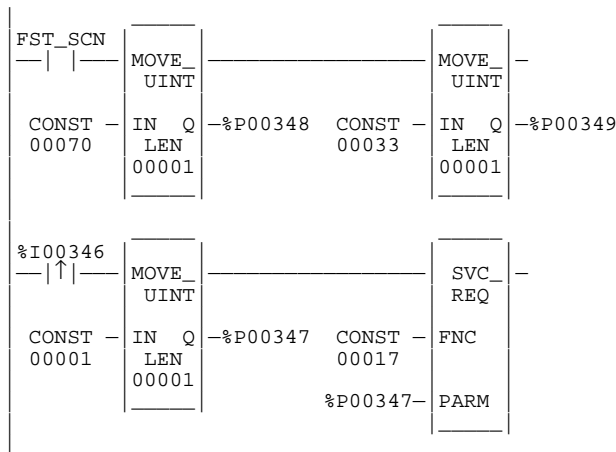
Example 2:

In the following example, when %T00001 transitions on, alarm interrupts from input %AI0006 are masked. The parameter block at %R00100 is set up on the first sweep.

```
 | FST_SCN    _____                      _____
 |—| |——————| MOVE_ |———————————————————| MOVE_ |—
 |          |  UINT |                   |  UINT |
 |          |       |                   |       |
 | CONST —| IN   Q |—%R00101   CONST —| IN   Q |—%R00102
 | 00010   | LEN   |            00006   | LEN   |
 |          | 00001 |                   | 00001 |
 |          |_____|                   |_____|
 |
 |
 | %T00001    _____                      _____
 |—|↑|——————| MOVE_ |———————————————————| SVC_  |—
 |          |  UINT |                   |  REQ  |
 |          |       |                   |       |
 | CONST —| IN   Q |—%R00100   CONST —| FNC   |
 | 00001   | LEN   |            00017   |       |
 |          | 00001 |                   |       |
 |          |_____|          %R00100—| PARM  |
 |                                      |_____|
 |
```
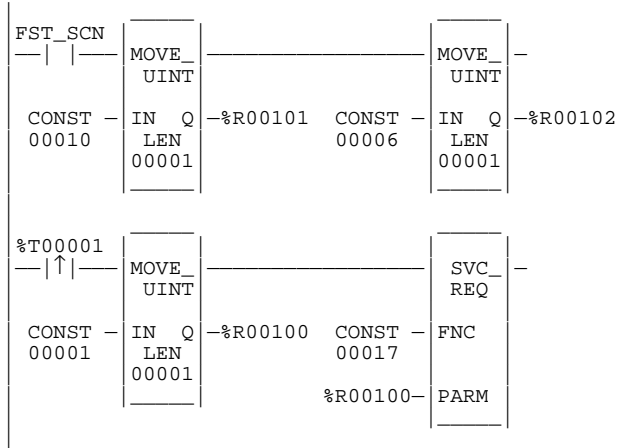
SVCREQ #18: Read I/O Override Status

Use SVCREQ function #18 in order to read the current status of %I and %Q overrides in the CPU.

The parameter block is an output parameter block only; it has a length of one word.

0 = No overrides are set.
1 = Overrides are set.

address

Example:

In the following example, the status of I/O overrides is always read into location %L01003. If any overrides are present, program block DISP_OR is called.

```
          |‾‾‾‾‾|                |‾‾‾‾‾|  ―
  ――――――――| SVC_|――――――――――――――――| EQ_ |
          | REQ |                | UINT|
          |     |                |     |            |‾‾‾‾‾‾‾‾|
  CONST ― | FNC |      CONST ― | I1  Q|――――――――――――| DISP_OR|
  00018   |     |      00001   |     |            |        |
          |     |              |     |            |_____|
  %L01003―| PARM|      %L01003―| I2  |
          |_____|                |_____|
```
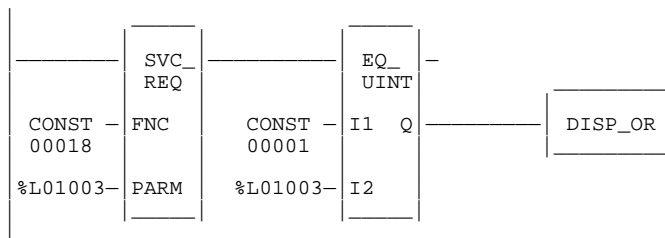
Note

SVCREQ #18 does not detect overrides in %G or %M memory types. Use %S0011 (OVR_PRE) to detect overrides in %I, %Q, %G, and/or %M memory types..

SVCREQ #19: Set Run Enable/Disable

Use SVCREQ function #19 to permit the ladder 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:

In the following example, when input %I00157 transitions to on, the **RUN DISABLE** mode is set. When the SVCREQ 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.

```
%I00157    _____                              _____                          %Q00157
—|↑|—— MOVE_ ——————————————————— SVC_ ———————————————————————— (S)—
            UINT                             REQ
                                                                 
 CONST —|IN   Q|—%R00100   CONST —|FNC
 00001   LEN             00019
         00001
         |_____|                  %R00100—|PARM
                                          |_____|


%Q00157    _____
—| |—— GT_  —
            UINT                  _____                          _____      %Q00158
                                                                            ——(R)—
%R00099—|I1   Q|——————— MOVE_ ——————————————————— SVC_ ——————————
                                 UINT                            REQ
 CONST —|I2        CONST —|IN   Q|—%R00100 CONST —|FNC
 00000   |_____|  00002   LEN            00019
                         00001
                         |_____|                 %R00100—|PARM
                                                          |_____|
```
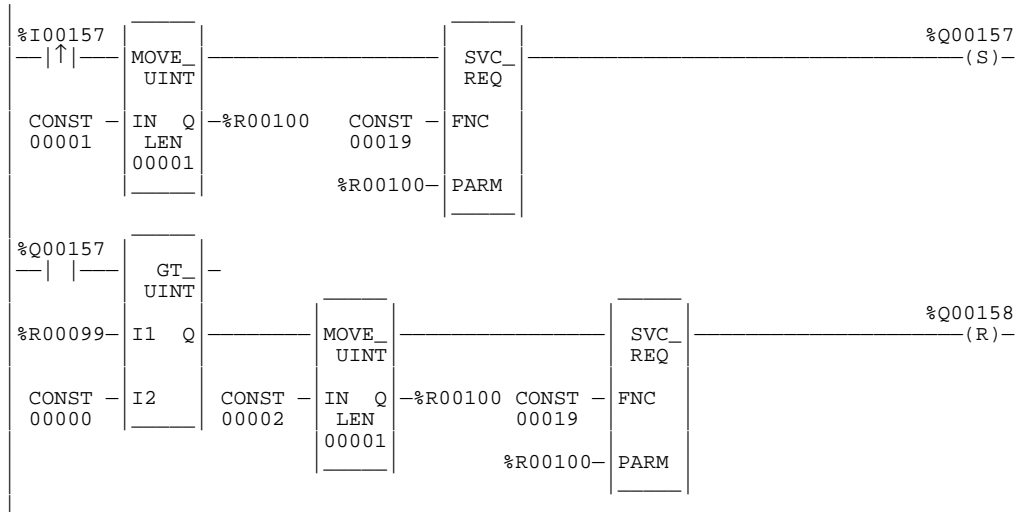
SVCREQ #20: Read Fault Tables

Use SVCREQ function #20 to retrieve the entire PLC or I/O fault table and return it to the ladder 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 PLC Fault Table (00h), Read I/O Fault Table (01h), Read Extended PLC 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 does returns only the fault table header information.

Input and Output Parameter Format for the Non-Extended Formats

The parameter block is an input and output parameter block. The Read PLC Fault Table (00h) and Read I/O Fault Table (01h), the input parameter block has the following format:

Address	00h = Read PLC fault table.
	01h = Read I/O fault table.
Address + 1	Always zero (0).

The output parameter block has this format:

Address	0 = PLC fault table
	1 = I/O fault table
Address + 1	Always zero (0)
Address + 2 through Address + 14	Unused
Address + 15 Address + 16 Address + 17	Time since last clear
Address + 18	Number of faults since last clear
Address + 19	Number of faults in queue
Address + 20	Number of faults read
Address + 21	Start of fault data

For the non-extended formats, each fault table entry is 21 words long (42 bytes). There are a maximum of 16 PLC fault table entries and 32 I/O fault table entries. If the fault table read is empty, no data is returned.

Note

SVCREQ #20 will not work unless there are 693 registers available.

Input and Output Parameter Format for the Extended Formats

The parameter block is an input and output parameter block. The Read Extended PLC Fault Table (80h) and Read Extended I/O Fault Table (81h), the input parameter block has the following format:

Address	80h = Read extended PLC 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.

The output parameter block has this format:

Address	80h = Extended PLC fault table
	81h = Extended I/O fault table
Address + 1	Starting index of faults to be read.
Address + 2 through Address + 14	Unused
Address + 15 Address + 16 Address + 17	Time since last clear
Address + 18	Number of faults since last clear
Address + 19	Number of faults in queue
Address + 20	Number of faults read
Address + 22	PLC name
Address + 23	
Address + 24	
Address + 25	
Address + 26	
Address + 27	
Address + 28	
Address + 29	
Address + 30	
Address + 31	
Address + 32	
Address + 33	
Address + 34	
Address + 35	
Address + 36	
Address + 37	Start of fault data

For Read Extended PLC Fault Table (80h) and Read Extended I/O Fault Table (81h), each extended fault table entry is 23 words long (46 bytes). There are a maximum of 40 PLC fault table entries and 40 I/O fault table entries. The default values are 16 PLC fault table entries and 32 I/O fault table entries. If the fault table read is empty, no data is returned.

Note

For the non-extended format, SVCREQ #20 will not work unless there are 693 consecutive registers available (beginning with the starting point). For the extended format, the PLC calculates the number of entries being read and returns an error if not enough registers.

The total size of the fault table for the extended fault format is

Header Size + ((# fault entries) * (size of fault entry))

The following table shows the return format of both a PLC fault table entry and an I/O fault table entry.

Address	PLC Fault Table	I/O Fault Table
Address + 21	Long/Short	Long/Short
Address + 22	Spare	Reference Address
Address + 23	PLC fault address	I/O fault address
Address + 24	PLC fault address	I/O fault address
Address + 25	Fault group and action	I/O fault address
Address + 26	Error code	Fault group and action
Address + 27	Fault extra data	Fault category and type
Address + 28	Fault extra data	Fault description
Address + 29	Fault extra data	Fault specific data
Address + 30	Fault extra data	Fault specific data
Address + 31	Fault extra data	Fault specific data
Address + 32	Fault extra data	Fault specific data
Address + 33	Fault extra data	Fault specific data
Address + 34	Fault extra data	Fault specific data
Address + 35	Fault extra data	Fault specific data
Address + 36	Fault extra data	Fault specific data
Address + 37	Fault extra data	Fault specific data
Address + 38	Fault extra data	Fault specific data
Address + 39	Time stamp	Time stamp
Address + 40	Time stamp	Time stamp
Address + 41	Time stamp	Time stamp

The Long/Short indicator in the first byte of Address + 21 defines the quantity of fault data present in the fault entry. In the PLC fault table, a long/short value of 00 represents 8 bytes of fault extra data present in the fault entry, and 01 represents 24 bytes of fault extra data. In the I/O fault table, 02 represents 5 bytes of fault specific data, and 03 represents 21 bytes.

For an explanation of each field, refer to Appendix B, "Interpreting Fault Tables Using Logicmaster 90-70 Software."

Example:

In the example below, when input %M00033 transitions on, the PLC fault table is read. When %M00034 transitions on, the I/O fault table is read. The parameter block is located at %R00500. When the SVCREQ function successfully executes, coil %M00035 is turned on.

```
|
|%M00033        _____                              _____                      %M00035
|—| |———————| MOVE_ |——————————+            ———| SVC_ |————————————————————————( S )—
|         | UINT |          |               | REQ |
|         |      |          |               |     |
|  CONST —| IN  Q|—%R00500   CONST —| FNC |
|  00000  | LEN  |                  00020  |     |
|         |00001 |                         |     |
|         |_____|           %R00500—| PARM |
|                                            |_____|
|
|%M00034        _____
|—| |———————| MOVE_ |——————————+
|         | UINT |          |
|         |      |          |
|  CONST —| IN  Q|—%R00500
|  00001  | LEN  |
|         |00001 |
|         |_____|
|
```
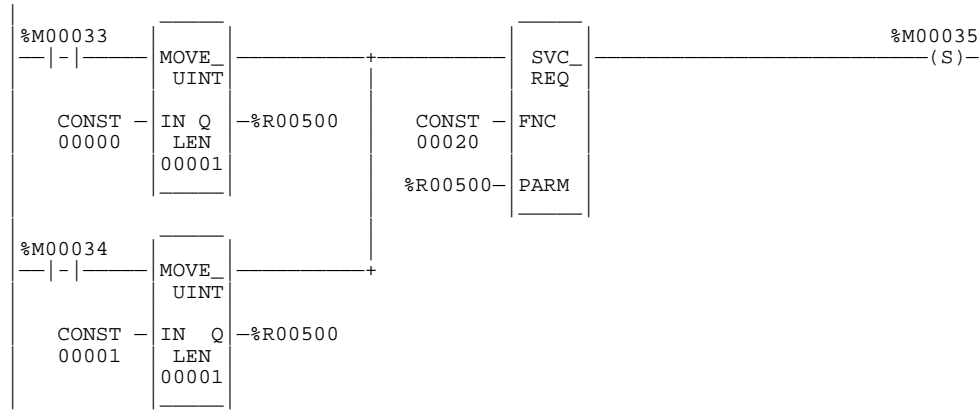
Example Extended Format:

In the example below, when input %M00033 transitions on, the Extended PLC fault table is read. The parameter block is located at %R00500. %R00500 contains the fault table type (PLC Extended); %R00501 contains the starting fault to read, and %R005002 contains the number of faults to read starting with the fault number in %R00501. When the SVCREQ function successfully executes, coil %M00034 is turned on.

```
|
|%M00033       _____                      _____                      _____
|—|↑|————————| MOVE_|——————————————————| MOVE_|——————————————————| MOVE_|——————————————( + )——
|        | WORD |                  | WORD |                  | WORD |
|        |      |                  |      |                  |      |
| CONST —| IN  Q|—%R00500   CONST —| IN  Q|—%R00501   CONST —| IN  Q|
| 00080  | LEN  |           00001  | LEN  |           00010  | LEN  |
|        |00001 |                  |00001 |                  |00001 |
|        |_____|                  |_____|                  |_____|
|
|%M00033       _____                                                        %M00034
|—| |————————| SVC__|————————————————————————————————————————————————————————( S )—
|        | REQ  |
|        |      |
| CONST —| FNC  |
| 00020  |      |
|        |      |
|%R00500—| PARM |
|        |_____|
|
```
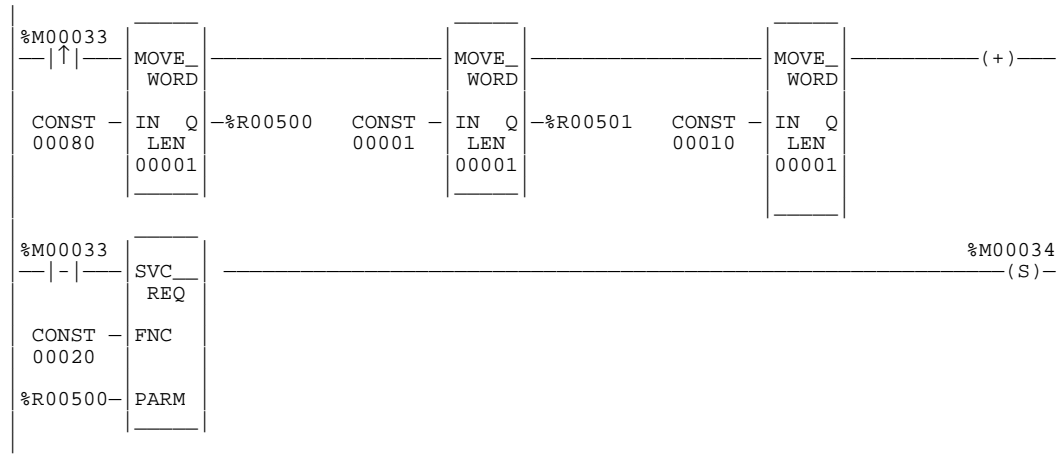
SVCREQ #21: User-Defined Fault Logging

Use SVCREQ function #21 to define a fault that can be displayed in the PLC 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 in order for an "Application Msg:" to be displayed. If the error code is in the range 81 to 112 decimal the PLC 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
Error 0 - 80	No bit set.
Error 81 - 112	Sets %SA.
Error 113 - 2047	No bit set.
Error 2048 – 32,767	Reserved.

When EN is active, the fault data array referenced by IN is logged as a fault to the PLC 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:

Parameter	MSB	LSB
Address	Error Code	
Address + 1	Text 2	Text 1
Address + 2	Text 4	Text 3
Address + 3	Text 6	Text 5
Address + 4	Text 8	Text 7
Address + 5	Text 10	Text 9
Address + 6	Text 12	Text 11
Address + 7	Text 14	Text 13
Address + 8	Text 16	Text 15
Address + 9	Text 18	Text 17
Address + 10	Text 20	Text 19
Address + 11	Text 22	Text 21
Address + 12	Text 24	Text 23

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, it will be converted to error code 0.

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 can be displayed by using the CTRL-F display of fault data in the Logicmaster 90-70 programmer PLC fault display.

For more information, refer to chapter 5, "PLC Control and Status," in the *Programming Software User's Manual*, GFK-0263.

Example:

In the following example, the value passed to IN1 is the fault error code. The value passed in, 16x0057, represents an error code of 87 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 which terminates the string. Thus, the decoding yields the string message **E_STOP ON**.

```
 |                                                                        
 | FST_EXE   _____                                                %Q00001
 |——| |——   |         |                                            _____
 |          | BLKMV   |————————————————————————————————————————————(  )—
 |          |  WORD   |
 |          |         |
 |  CONST — | IN1   Q |—%P00001
 |   0057   |         |
 |          |         |
 |  CONST — | IN2     |
 |   2D45   |         |
 |          |         |
 |  CONST — | IN3     |
 |   5453   |         |
 |          |         |
 |  CONST — | IN4     |
 |   504F   |         |
 |          |         |
 |  CONST — | IN5     |
 |   4F20   |         |
 |          |         |
 |  CONST — | IN6     |
 |   004E   |         |
 |          |         |
 |  CONST — | IN7     |
 |          |_____|
 |
 |           _____
 |%I00050   |       |
 |——| |——   | SVC_  |—
 |          |  REQ  |
 |          |       |
 |  CONST — | FNC   |
 |  00021   |       |
 |          |       |
 |%P00001—  | PARM  |
 |          |_____|
 |
```

SVCREQ #22: Mask/Unmask Timed Interrupts

Use SVCREQ function #22 to mask or unmask timed interrupts and to read the current mask. When the interrupts are masked, the PLC CPU will 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 will occur 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:

0 = Read interrupt mask.	address

The PLC returns this format:

0 = Read interrupt mask.	address
0 = Timed interrupts are unmasked. 1 = Timed interrupts are masked.	address + 1

To change the current mask, use this format:

1 = Mask/unmask interrupts.	address
0 = Unmask timed interrupts. 1 = Mask timed interrupts.	address + 1

Example:

In the following example, when input %I00055 transitions on, timed interrupts are masked.

```
 _____                    _____                    _____
%I00055 |      |                  |      |                  |      |
—| ↑ |——| MOVE_ |——————————————————| MOVE_ |——————————————————| SVC_ |
        | UINT |                  | UINT |                  | REQ  |
        |      |                  |      |                  |      |
CONST —| IN  Q |—%R01002  CONST —| FN  Q |—%R01003  CONST —| FNC  |
00001  | LEN  |          00001  | LEN  |          00022  |      |
       | 00001 |                 | 00001 |                 |      |
       |_____|                 |_____|         %R01002—| PARM |
                                                         |_____|
```
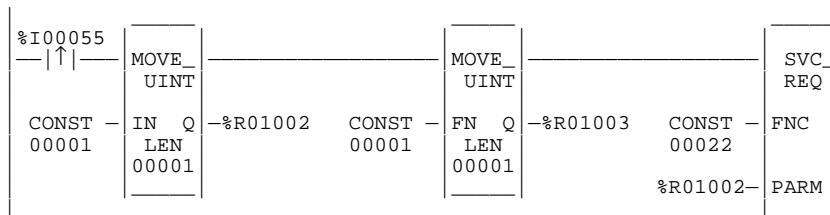
SVCREQ #23: Read Master Checksum

SVCREQ function #23 returns master checksums for the set of user program(s) and the configuration. It also returns the checksum for the block from which the service request is made.

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.

There is no input parameter block for this service request. The output parameter block layout is as follows; it requires 15 words of memory.

Output Parameter Block		**Word Address**
Program Checksum Valid (0 = not valid, 1 = valid)		address
Master Program Checksum Valid (0 = not valid, 1 = valid)		address + 1
Master Configuration Checksum Valid (0 = not valid, 1 = valid)		address + 2
Number of LD/SFC Blocks (including _MAIN)		address + 3
Size of User Program in bytes (DWORD data type)		address + 4
Program Set Additive Checksum		address + 6
Program CRC Checksum (DWORD data type)		address + 7
Size of Configuration Data in Bytes		address + 9
Configuration Additive Checksum		address + 10
Configuration CRC Checksum (DWORD data type)		address + 11
Always zero (high byte)	Currently Executing Block's Additive Checksum	address + 13
Currently Executing Block's CRC Checksum		address + 14

Example:

In the following example, 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.

```
 | << RUNG 5 >>
 |
 |                 _____
 |%M00054         |      |                                                          %M00054
 |—| / |——|  TMR  |———————————————————————————————————————————————————————————————( )—
 |                | 1.00s|
 |                |      |
 | CONST —|PV  CV|
 | 00060          |_____|
 |                 %P00013
 |
 | << RUNG 6 >>
 |
 |                 _____                                    _____
 |%M00054         |      |                                  |      |
 |—|  |——————|  SVC_ |——————————————————————————————|  NE_ |
 |                |  REQ |                                  | DINT |
 |                |      |                                  |      |         %M00055
 |       CONST —|FNC   |                      %P00022—|I1  Q|————————————————(SM)—
 |       00023    |      |                      %P00031—|I2   |
 |                |      |                                  |_____|
 |                | PARM |
 |       %P00016—|_____|
 |
 | << RUNG 7 >>
 |
 |                 _____
 |%M00054         |      |
 |—|  |——————|MOVE_ |
 |                | DWORD|
 |                |      |
 |       %P00022—|IN  Q|—%P00031
 |                | LEN  |
 |                | 00001|
 |                |_____|
 |
```
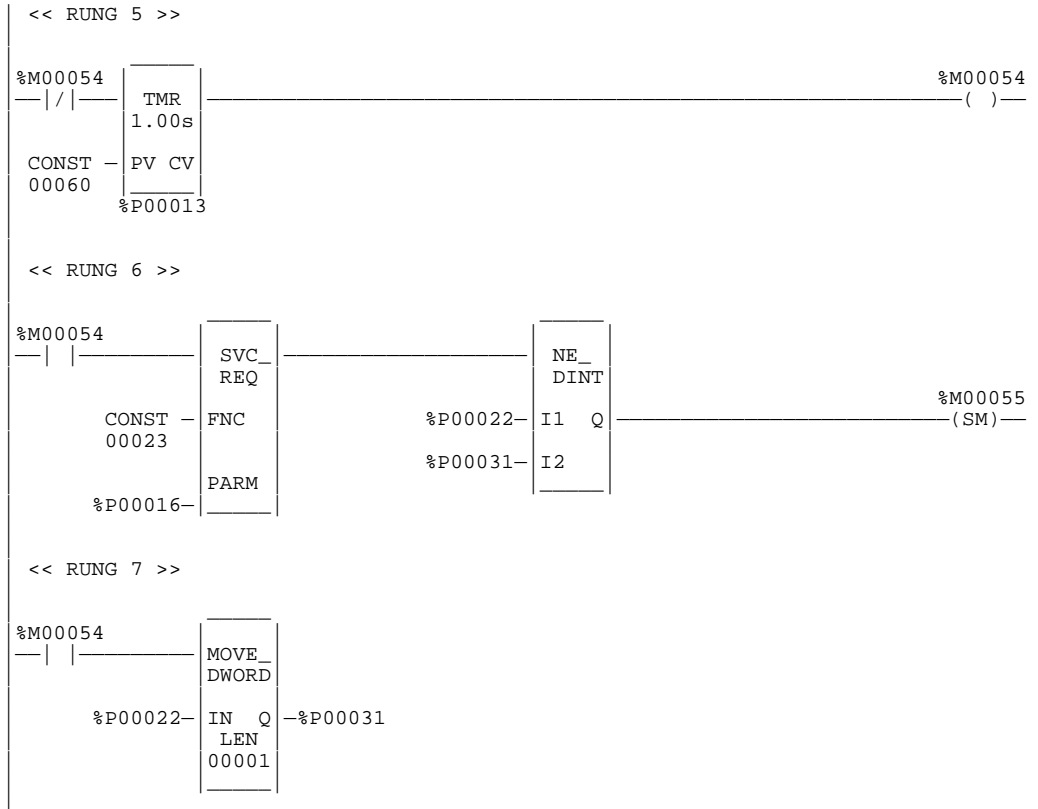
SVCREQ #25: Disable/Enable EXE Block and Standalone C Program Checksums

Use SVCREQ function #25 to enable or disable the inclusion of EXE blocks and standalone C programs (i.e., C applications) in the background checksum calculation. The default is to include the checksums.

This service request uses only an input parameter block.

0 = Disable C applications inclusion in checksum calculation. 1 = Enable C application inclusion in checksum calculation.	address

The parameter block is unchanged after execution of the service request.

Example:

In the following example, when the coil TEST transitions from OFF to ON, SVCREQ #25 executes to disable the inclusion of EXE blocks in the background checksum calculation. When coil TEST transitions from ON to OFF, the SVCREQ executes to again include EXE blocks in the background checksum calculation.

```
   TEST          _____                                            _____
 ──|↑|────────| MOVE_ |──────────+─────────────────────────────| SVC_ |──────
             |  UINT |                                           |  REQ |
             |      |                                            |      |
   CONST ───| IN  Q |─%R00150                          CONST ───| FNC  |
   00000     | LEN  |                                  00025     |      |
             | 00001|                                            |      |
             |_____|                              %R00150───| PARM |
                                                               |_____|

   TEST          _____
 ──|↓|────────| MOVE_ |──────────+
             |  UINT |
             |      |
   CONST ───| IN  Q |─%R00150
   00001     | LEN  |
             | 00001|
             |_____|
```
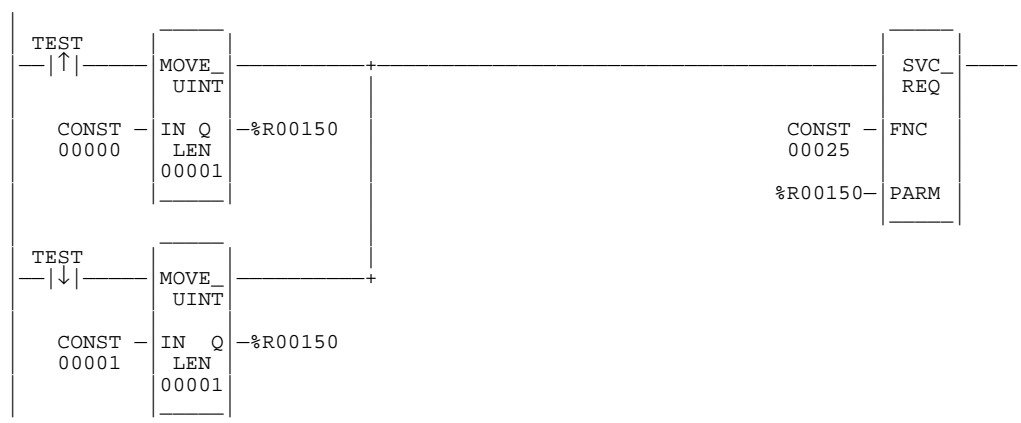
SVCREQ #26: Role Switch

Note

SVCREQ #26 is intended for use with Hot Standby CPU Redundancy which is only available on Model 780 CPUs (IC697CPU780), and with Enhanced Hot Standby CPU Redundancy on Models IC697CGR772 and IC697CGR935. For more information about Hot Standby CPU Redundancy, refer to the *Series 90-70 Hot Standby CPU Redundancy User's Guide*, GFK-0827. For more information about Enhanced Hot Standby CPU Redundancy, refer to the *Series 90-70 Enhanced Hot Standby CPU Redundancy User's Guide*, GFK-1527.

Use SVCREQ function #26 to cause the CPUs to switch roles on the next sweep (active to backup and backup to active) if the CPUs are synchronized and the timing requirements of the role switch request are met. A manual role switch cannot occur within 10 seconds of a previous manual role switch. Role switches due to failures or resynchronization are always allowed (the 10 second limitation does not apply).

Note

Power flow from SVCREQ #26 indicates that a role switch will be attempted on the next sweep. It does **not** indicate that a role switch has occurred or that a role switch will occur on the next sweep.

This function has no associated parameter block; however, the programming software requires that an entry be made for PARM. Enter any appropriate reference here; it will not be used.

Example:

In the following example, a switch on a control console is wired to %I00001, the input to the SVCREQ #26 function block. When closed, the switch will activate the SVCREQ #26, causing a role switch between CPUs.

```
                _____
|%I00001        |     |                                                    %M00001
|—| ↑ |——       | SVC_|  _____ ( )—
|               | REQ |
|               |     |
| CONST —       | FNC |
| 00026         |     |
|               |     |
|%R00001—       | PARM|
|               |_____|
```
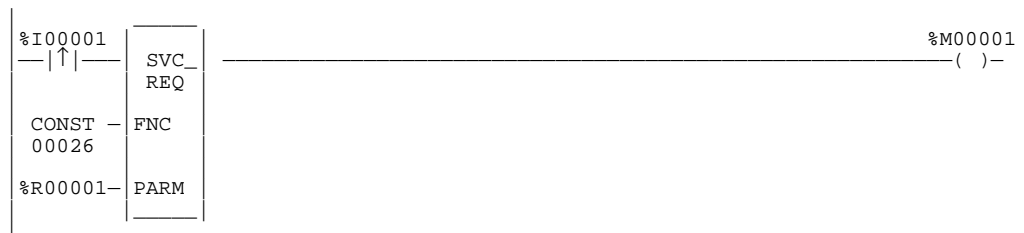
The 10-second limitation allows this SVCREQ to be in both CPUs so that only a single switch occurs if the input is seen by both CPUs.

SVCREQ #27 and #28: Write to/Read from Reverse Transfer Area

SVCREQ #27 and #28 are intended for use with Hot Standby CPU Redundancy *available only* on IC697CPU780 and IC697CGR935 CPUs. Refer to "Programming a Data Transfer from Backup Unit to Active Unit" in the *Series 90™-70 Enhanced Hot Standby CPU Redundancy User's Guide* (GFK-1527) for all information about using this service request.

SVCREQ #32: Suspend/Resume I/O Interrupt

Use SVCREQ function #32 to suspend a set of I/O interrupts and cause occurrences of these interrupts to be enqueued until these interrupts are resumed. The set of I/O interrupts are those that can be generated from the 90-70 High Speed Counter. The number of I/O interrupts that can be enqueued depends on the I/O module's capabilities. The PLC 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 should be suspended and resumed within a single sweep.

This service request uses only an input parameter block. Its length is three words.

0 = resume interrupt. 1 = suspend interrupt.	address
memory type	address + 1
reference (offset)	address + 2

Successful execution will occur 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 module must be a 90-70 High Speed Counter.)

- The reference address specified is not the first %I reference for the High Speed Counter.

- Communication between the PLC CPU and this I/O module has failed. (The board is not present, or it has experienced a fatal fault.)

Example:

In the following example, 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 3400. (%AI00001 is the first nondiscrete 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.

Example

```
     FST_SCN  ┌──────┐                        ┌──────┐
     ─┤ ├──── │MOVE_ │────────────────────────│MOVE_ │─
              │ INT  │                        │ INT  │
              │      │                        │      │
      CONST ─ │IN   Q│─%P00002      CONST ─   │IN   Q│─%P00003
      +00070  │LEN   │              +00065    │LEN   │
              │00001 │                        │00001 │
              └──────┘                        └──────┘


              ┌──────┐                        ┌──────┐                              %T00001
     ──────── │MOVE_ │────────────────────────│ SVC_ │──────────────────────────────( )──
              │ INT  │                        │ REQ  │
              │      │                        │      │
      CONST ─ │IN   Q│─%P00001      CONST ─   │FNC   │
      +00001  │LEN   │                00032   │      │
              │00001 │                        │      │
              │      │              %P00001─  │PARM  │
              └──────┘                        └──────┘



     %T00001  ┌──────┐
     ──┤ ├──── │ EQ_  │─
              │ INT  │
              │      │                        ┌──────┐                              %T00006
      %AI0001─│I1   Q│────────────────────────│MOVE_ │──────────────────────────────( )──
              │      │                        │ INT  │
              │      │                        │      │
      CONST _ │      │      %AI0001 ─         │IN   Q│─%R00001
      +03400  └──────┘                        │LEN   │
                                              │00001 │
                                              └──────┘



              ┌──────┐                        ┌──────┐
     ──────── │MOVE_ │────────────────────────│ SVC_ │─
              │ INT  │                        │ REQ  │
              │      │                        │      │
      CONST ─ │IN   Q│─%P00001      CONST ─   │FNC   │
      +00000  │LEN   │                00032   │      │
              │00001 │                        │      │
              │      │              %P00001─  │PARM  │
              └──────┘                        └──────┘


     .
```
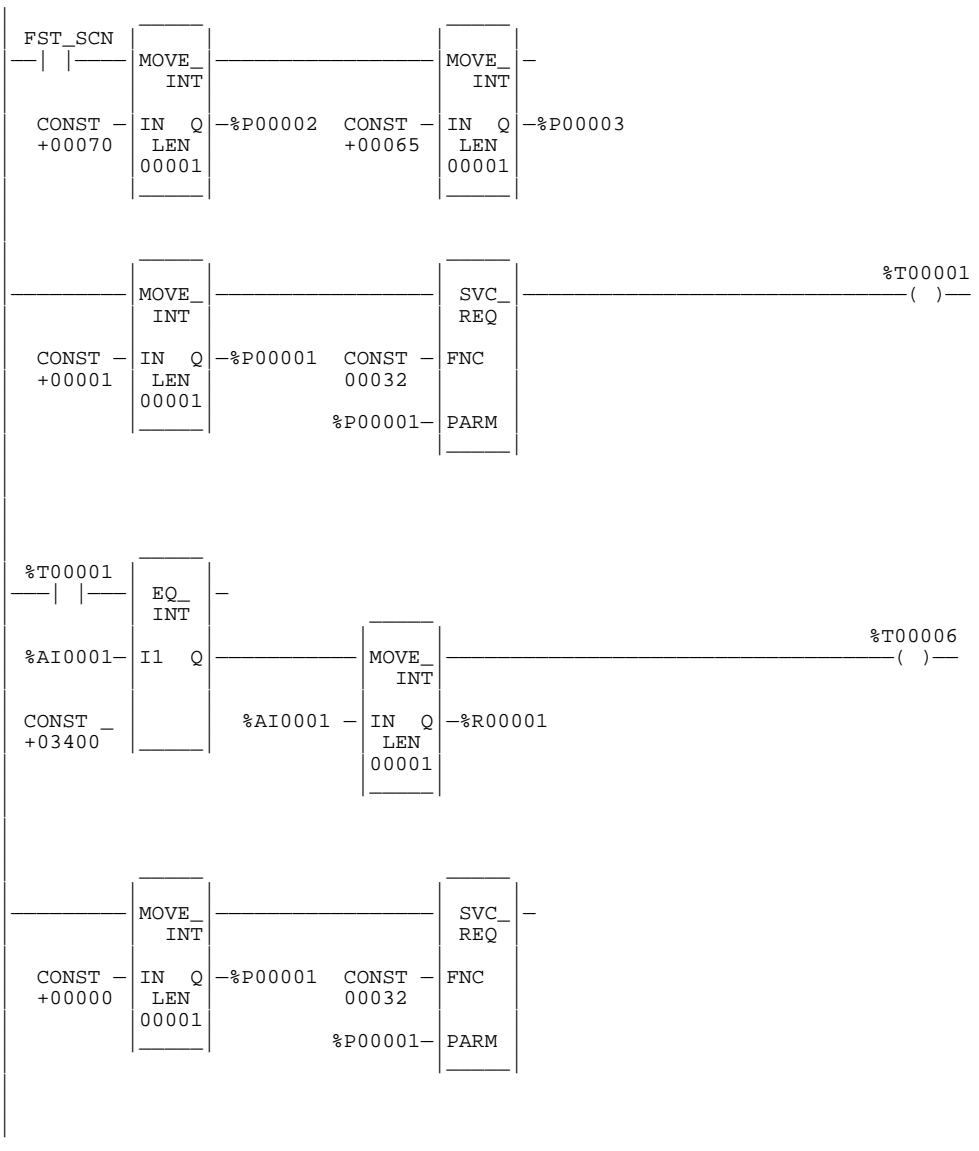
SVCREQ #39: ESCM Port Status

Use Service Request #39 ESCM (Embedded Serial Comm Module) Port Status to provide status of Ports 1 and 2 to the PLC CPU. The address specified in the parameter should be set to a value of 1 or 2 (indicating Port 1 or 2). The port status will be returned in the following value.

1 = Port 1 2 = Port 2	address
The location in which the status is returned	address + 1

Successful execution will occur unless:

- The first parameter is not 1 or 2.

- The CPU does not support use of Ports 1 and 2.

Note

Only CPX and CGR models of CPUs support Ports 1 and 2.

Example:

In the following example, when Service Request # 39 executes, the CPU will determine the port number from the command block word (%M00001 in the example shown below), then validate the port number. It will then retrieve the current corresponding ESCM status word and place the retrieved status word into the %M00017.

If the port number is invalid, or if input power flow is off, or if the current CPU does not support the use of Ports 1 and 2, then output power will be OFF. If the port is valid and the CPU does support the use of Ports 1 and 2, then output power will be ON.

```
  %I00055   _____                                             _____                        %T00001
 —| ↑ |——| MOVE_ |—————————————————————————————————————————| SVC_ |—————————————————————————( )——
          | WORD |                                           | REQ  |
          |      |                                           |      |
  CONST —|IN   Q|—%M00001                           CONST —|FNC   |
  00001   | LEN  |                                    00039   |      |
          | 00001|                                           |      |
          |_____|                             %M00001—|PARM  |
                                                             |_____|
```
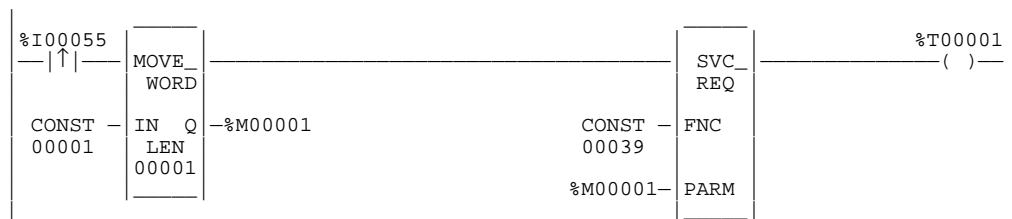
Return Values

The ESCM status information is delivered in the form of a word of bit-encoded data. Each bit of the first four bits (beginning with the Least Significant Bit) indicates something about the status of the ESCM as described in this table shown below:

State	Bit Position	Description
PORTN_OK	0	Requested port is ready. If value is 1, the port is ready. If value is 0, the port is not usable.
PORTN_ACTIVE	1	There is activity on this port. If value is 1, the port is active. If value is 0, the port is inactive.
PORTN_DISABLED	2	Requested port is disabled. If value is 1, the port is disabled. If value is 0, the port is enabled.
PORTN_FUSE_BLOWN	3	Requested port's fuse is blown (for Port 2) or supply voltage is not within range (for Port 1). If value is 1, the fuse is blown (or voltage not within range). If value is 0, the fuse (or supply voltage) is okay.
RESERVED	4–16	Set to zero (reserved).

SVCREQ #44: Logic Driven Dynamic Ethernet Global Data

Logic Driven Dynamic Ethernet Global Data (EGD), available on release 7.91andlater CPX CPUs, allows you to dynamically establish and terminate Ethernet Global Data exchanges from within the logic program. This feature uses SVCREQ #44 to establish, terminate, and monitor the exchanges. Hardware configuration of Ethernet Global Data is not required; however, the adapter(s) must be configured by the Configuration software.

The adapter module IC697CMM742, version 2.70 or later supports Logic Driven EGD.

A maximum number of 32 Logic Driven exchanges are allowed per PLC. A total of 255 exchanges is possible. Each Logic Driven exchange can have a maximum of 8 variables per exchange.

SVCREQ #44 must not be issued (that is, from an interrupt block) while another SVCREQ #44 is in process in the same CPU.

Note

On the sweep that an exchange is established, the CPU will require more time during the start of sweep processing. Furthermore, if you are starting multiple LD EGD connections during a single sweep, the watchdog timeout may need to be adjusted.

Service Request Function Block

Table 12-2. General Format of SVCREQ #44 Function Block

Status	Address
Indicates success or failure of request. Consult the set of possible responses for each command. The user should set this field to an initial value of 0.	
Command	Address + 1
Indicates which command is to be executed: **1** Set local producer ID **2** Retrieve local producer ID **3** Establish a produced exchange **4** Establish a consumed exchange **5** Terminate produced exchange **6** Terminate consumed exchange **7** Refresh production data every sweep	
Command Specific Information Block	Address + 2
The format of the remaining fields depends on the command selected. See the descriptions below for each command.	. . . Address + n

Returned Status Values

The following general errors can be returned in the status. Other values are possible for each command.

Table 12-3. Command Status Possible for All Commands

Value	Description	Power Flow
-18	Invalid command. Not in range of 1 – 7.	No
-19	Parameter Block length exceeds memory range.	No
-23	Service Request #44 power flow interrupted. That is, a new command was issued before the last one completed.	No

Details of the Service Request Commands

Table 12-2 lists seven commands that this service request can initiate. Each of these is described in detail in this section.

Command 1 - Set Local Producer ID

A Local Producer ID is used to identify this PLC to other exchange recipients and providers. The Local Producer must exist and have a non-zero value before an exchange can be created. There are two ways to create the producer ID: 1) Configure the ID using Control software or 2) Use the Set Local Producer ID command. Once set to a non-zero value, the local producer ID cannot be changed by the logic.

The Local Producer ID is specified in four words. Each word typically contains a portion of the dot separated IP Address. When this command is used to assign the Local Producer ID, it is only necessary that one of the 4 words contain a non-zero value; however, make sure that each PLC is assigned a unique value. Setting the ID is not allowed if hardware configuration has not been stored to the PLC CPU.

If the Local Producer ID is set using this command, then it will be reset to 0.0.0.0 when the CPU transitions to STOP or power is cycled on the CPU.

Table 12-4. Format of Set Local Producer ID Command

	Address
Status See table 12-5. The user should initially set this field to 0.	Address
Set Local Producer ID Command Always 1.	Address + 1
Local Producer ID Four words containing the four parts of the producer ID. Each word can have a value of 0 to 255. At least one word must be non-zero.	Address + 2 to Address + 5

The Command Status is returned upon the completion of the Set service request as detailed in the following table.

Table 12-5. Command Status for Set Local Producer ID Command

Value	Description	Power Flow
1	ID is set as requested or the ID specified was the same as existing ID.	Yes
-2	Requested ID is different from value stored in the downloaded configuration.	No
-3	Requested ID is different from value set by a previous *set-producer ID* service request.	No
-11	No hardware configuration stored.	No
-16	Invalid Local Producer ID (0 or out of range)	No

Command 2 - Retrieve Local Producer ID

This command is useful when the ID has been set using Control software. If the command returns 0.0.0.0, the local producer ID either does not exist or has a 0 value and must be set to a non-zero value to use logic driven exchanges.

Table 12-6. Format of Retrieve Local Producer ID Command

	Address
Status See Table 12-7. The user should initially set this field to 0.	Address
Retrieve Local Producer ID Command Always 2.	Address + 1
Local Producer ID Four words containing the four parts of the current producer ID in the CPU. Each word returned will be in the range of 0 to 255.	Address + 2 to Address + 5

The command status is returned upon the completion of the Retrieve Local Producer ID command as detailed in the following table.

Table 12-7. Command Status for Retrieve Local Producer ID Command

Value	Description	Power Flow
1	ID configured by Set Local Producer ID.	Yes
2	ID was configured in the stored configuration.	Yes
-1	Local producer ID is not set.	No

Commands 3 and 4 - Establish a Produced Exchange/Consumed Exchange

The creation of an exchange will require multiple sweeps of the program. However, Power Flow will be passed on the calling sweep if all of the following conditions are met:

- The local Producer ID is not 0.0.0.0.

- The exchange to be established (identified by Producer ID and Exchange ID) does not already exist.

- Variable addresses are valid and the number of variables specified does not exceed the maximum number allowed (8).

- Creation of the exchange does not exceed the maximum number of logic driven (32) or total number of exchanges allowed (255).

- Sufficient User Memory is available to create the exchange.

If any of the above conditions fail, power flow will not be passed, and the command status will indicate the cause of failure. The exchange will not be created and the exchange status word will not be altered.

If power flow is passed, the exchange has been created, but the exchange is not yet ready for use by the user program. The command status will be set to 1 and the exchange status word will be set to a value of IPLC_CREATE_IN_PROGRESS (24). The user program must monitor the exchange status word on subsequent sweeps to determine when the Exchange Creation has completed (changed from IPLC_CREATE_IN_PROGRESS). Up to 12 seconds may elapse before the exchange status word is set to a failure or completion status.

If the exchange status word is set to one of the values in Table 10, the exchange has not been created successfully and the local exchange definition is deleted. For a complete list of possible Exchange Status Word values for Logic Driven exchanges, refer to Table 12-19.

Table 12-8. Exchange Status Word for Establish Exchange Commands

Exchange Status Word	Description
12	Insufficient dual port memory in Adapter Module.
18	Adapter Module has failed.
26	Adapter Module did not respond to the request within 6 seconds.
28	Other failure.

Exchanges can be established in parallel. The user program may begin creation of all exchanges in the same sweep, if desired. Any subsequent commands related to an exchange that has not successfully completed creation will fail and must be reissued when creation has completed.

A maximum of eight variables is allowed in each exchange.

For each Logic Driven exchange, the amount of PLC user memory required will be allocated when the exchange is established. The memory used for Logic Driven exchanges counts against the total memory available for Configuration and Logic. Approximately $176 + N*16$ bytes will be charged for each exchange, where N is the number of variables used in that exchange. The cumulative

amount of user memory used for Logic Driven exchanges, as well as the amount of user memory remaining, will not be visible when using the current versions of Logicmaster 90-70.

Use of Logic-driven EGD requires that sufficient time be given to the System Communications window. If the System Communications window does not run, then the establishment of Logic Driven EGD exchanges is expected to fail with the exchange status word set to 26.

The format for creating produced and consumed exchanges differs. This subject is covered in the following sections.

Format for the Establish a Produced Exchange Command

The format for the Establish a Produced Exchange command is as follows.

Table 12-9. Format of the Establish a Produced Exchange Command

	Address
Status See Table 12-10. The user should initially set this field to 0.	Address
Establish Produced Exchange Command Always 3.	Address + 1
Reserved Four words. Must be set to 0	Address + 2 to Address + 5
Exchange ID Allowable range: 32769 to 49151.	Address + 6
Reserved Two words. Must be set to 0	Address + 7 to Address + 8
Rack/Slot of Adapter Module Upper Byte = Rack Lower Byte = Slot	Address + 9
Production Period 10 through 3,600,000 in units of milliseconds (2 words)	Address + 10 to Address + 11
Exchange Status Word Location Four words. Refer to "Exchange Status Word" on page 12-84 for the format of this address. The NULL selector may not be used.	Address + 12 to Address + 15
Destination Address Type 1 = IP address 2 = Name String (must end in 0) 3 = Multicast Group ID	Address + 16
Destination Address Value 16 words	Address + 17 to Address + 32
Number of Variables 1 to 8	Address + 33
Configuration Data Six words per variable. Refer to "Variables Address Definition" on page 12-86 for the format of this address.	Address + 34 to …

Possible status values for the Establish a Produced Exchange command are shown below.

Table 12-10. Command Status for the Establish a Produced Exchange Command

Value	Description	Power Flow
1	Exchange information accepted.	Yes
-1	Local producer ID is not set.	No
-5	Exchange ID out of range.	No
-6	Exchange already exists.	No
-7	Too many exchanges.	No
-8	Too many variables.	No
-9	Maximum variable size exceeded.	No
-10	Insufficient resources available	No
-12	The rack/slot location is not properly configured, or does not support EGD.	No
-13	Rack/Slot is invalid or out of range.	No
-14	Status Word Address is unassigned or invalid.	No
-16	Reserved Words (3—6) invalid. (Must be 0.)	No
-22	Invalid Address Type. Must be in range of 1—3.	No
-24	Invalid value specified for the destination address.	No
-101 to -134	Parameter at word [Value] – 101 is incorrect.	No
-135 to -177	Variable assignment beginning at word [Value] – 101 is invalid.	No

The following rules apply to produced exchanges:

- Duplicate produced exchanges within the same PLC are not allowed. If a produced exchange with the same Exchange ID already exists, the request will be rejected.

- Data Production begins as soon as the exchange creation is completed successfully (may take several sweeps).

Format for the Establish a Consumed Exchange Command

The format for the Establish a Consumed Exchange command is as follows.

Table 12-11. Format of the Establish a Consumed Exchange Command

	Address
Status See Table 12-12. The user should initially set this field to 0.	Address
Establish Consumed Exchange Command Always 4.	Address + 1
Producer ID Four words. Must not be 0.0.0.0. This is the ID of the producer of the exchange. Not the Local Producer ID of the current CPU.	Address + 2 to Address + 5
Exchange ID Allowable range: 32769 to 49151 for logic driven exchanges or 1 to 16383 for static exchanges.	Address + 6
Reserved Two words. Must be set to 0	Address + 7 to Address + 8
Rack/Slot of Adapter Module Upper Byte = Rack Lower Byte = Slot	Address + 9
Consumption Period 10 through 3,600,000 in units of milliseconds (2 words)	Address + 10 to Address + 11
Exchange Status Word Location Four words. Refer to "Exchange Status Word" on page 12-84 for the format of this address. The NULL selector may not be used.	Address + 12 to Address + 15
Group ID 0 if production is peer-to-peer 1 – 32 if a multicast group is used.	Address + 16
Update Timeout Period In units of milliseconds, must be 0 (not used) or set larger than the Consumption Period. The range is the same as that of the Consumption Period. *It is recommended that the timeout period be set at least 20 ms larger than the consumption period to prevent false timeouts.* (2 words)	Address + 17 to Address + 18
Timestamp Location Four words. The timestamp is used to record the time when the data was produced by the CPU. If the NULL selector is used in the location address, no timestamp information will be available. Refer to "Timestamp" on page 12-86 for the format of this address and the format of the timestamp located at that address.	Address + 19 to Address + 22
Reserved Ten words. Must be set to 0.	Address + 23 to Address + 32
Number of Variables 1 to 8	Address + 33
Configuration Data Six words per variable. Refer to "Variables Address Definition" on page 12-86 for the format of this address.	Address + 34 to …

Possible status values for the Establish a Consumed Exchange command are shown below.

Table 12-12. Command Status for the Establish a Consumed Exchange Command

Value	Description	Power Flow
1	Exchange information accepted.	Yes
-1	Local producer ID is not set.	No
-5	Exchange ID out of range.	No
-6	Exchange already exists.	No
-7	Too many exchanges.	No
-8	Too many variables.	No
-9	Maximum variable size exceeded.	No
-10	Insufficient resources available	No
-12	The rack/slot location is not properly configured, or does not support EGD.	No
-13	Rack/Slot is invalid or out of range.	No
-14	Status Word Address is unassigned or invalid.	No
-15	Timestamp Address is invalid.	No
-16	Producer ID is unassigned (zero) or out of range.	No
-17	Producer ID specified matches Local Producer ID.	No
-20	Consumption Period is out of range, must be 10ms – 3,600,000ms.	No
-21	The Update Timeout Period is invalid. Must be 0 (not used) or be greater than the Consumption Period. The range is the same as that of the Consumption Period.	No
-22	Invalid Group ID, must be 0 or in range of 1 – 32.	No
-101 to -134	Parameter at word [Value] – 101 is incorrect.	No
-135 to -177	Variable assignment beginning at word [Value] – 101 is invalid.	No

The following rules apply to establishing consumed exchanges:

- Duplicate exchanges within the same PLC are not allowed. If an exchange with the same Producer ID and Exchange ID already exists (static or dynamic, consumed or produced), the request will be rejected.

- Data Consumption begins as soon as the exchange is created successfully (may take several sweeps).

Commands 5 and 6 - Terminate a Produced Exchange/Consumed Exchange

Logic Driven EGD exchanges will be terminated automatically on a given CPU when that CPU transitions to STOP or is power-cycled. You can also terminate these exchanges explicitly by using the Terminate Produced Exchange or Terminate Consumed Exchange commands. These commands cannot terminate exchanges configured using Control software for this CPU.

Termination may require several sweeps to complete. If the exchange can be completed, a 1 will be returned as the command status and, at completion, the Exchange Status Word will be set to IPLC_EXCHANGE_DELETED (30). If the exchange cannot be deleted, an error status will be returned in the command status and the Exchange Status Word will not be affected. The format of these commands is shown below.

Table 12-13. Format of the Terminate Produced Exchange Command

	Address
Status See Table 12-15. This field should be initially set to 0.	Address
Terminate Produced Exchange Command Always 5.	Address + 1
Reserved Four words. Must be set to 0	Address + 2 to Address + 5
Exchange ID Allowable range: 32769 to 49151.	Address + 6
Reserved Must be set to 0	Address + 7

Table 12-14. Format of the Terminate Consumed Exchange Command

	Address
Status See Table 12-15. This field should be initially set to 0.	Address
Terminate Consumed Exchange Command Always 6.	Address + 1
Reserved Four words. Must be set to 0	Address + 2 to Address + 5
Producer ID Four words. Must match the ID used to create the exchange.	Address + 6
Exchange ID Allowable range: 32769 to 49151 for logic driven exchanges, or 1 to 16383 for static exchanges.	Address + 7
Reserved User must set to 0.	Address + 7

The following table shows the possible command statuses from the two terminate commands.

Table 12-15. Command Status for the Terminate Exchange Commands

Value	Description	Power Flow
1	Exchange deleted as requested.	Yes
-1	Local producer ID is not set.	No
-5	Exchange ID out of range.	No

-16	**Terminate Produced Exchange**
	    Reserved Words (3 through 6) must be 0.
	Terminate Consumed Exchange
	    Invalid Producer ID
		No

-108	Reserved word 8 must be zero.	No.

Command 7 - Refresh Production Data Every Sweep

By default, the production data is updated only once during the production period. The Refresh Production Data Every Sweep command causes the data for every Logic Driven production exchange to be updated every sweep of the CPU, regardless of the specified production period for the exchanges. The effect of this request cannot be cancelled except by putting the CPU into STOP mode or by CPU power-cycle. Static exchanges are not affected by this command nor do Static exchanges preclude using this command for Logic Driven Exchanges.

The Exchange Status Word will be updated every CPU sweep, but data on the wire will be produced as specified by the exchange's production period.

The Local Producer ID must be set (either by configuration or by the Set Local Producer ID command) prior to issuing the Refresh Production Data Every Sweep command. The Refresh Production Data Every Sweep command must be made prior to commands to establish logic driven exchanges (either produced or consumed).

Table 12-16. Format of the Refresh Production Data Every Sweep Command

	Address
Status See Table 12-16. This field should be initially set to 0.	Address
Refresh Production Data Every Sweep Command Always 7.	Address + 1
Reserved Two words. Must be set to 0	Address + 2 to Address + 3

The command status is returned upon the completion of the Refresh Production Data Every Sweep command as detailed in Table 12-17.

Table 12-17. Command Status for Refresh Production Data Every Sweep Command

Value	Description	Power Flow
1	Success.	Yes
-1	Local producer ID is not set.	No
-23	Service Request #44 power flow interrupted.	No
-25	Command is not valid in the context. The request must be made prior to any request to establish an exchange.	No
-103	Reserved word 3 of the parameter block is not 0.	No.
-104	Reserved word 4 of the parameter block is not 0.	No

Additional Notes on Logic Driven Dynamic Ethernet Global Data

Assigning Exchange IDs

For every exchange in the local PLC, the combination of Producer ID and Exchange ID must be unique regardless of whether it is static, dynamic, consumed or produced.

For Logic Driven produced exchanges, each exchange in the local PLC must have a unique Exchange ID. The available range for these Exchange IDs is 32769 through 49151. It is suggested that the customer use a convenient subset of the range (for example 40000 - 49151).

For Logic Driven Consumed Exchanges, the Producer ID and Exchange ID must be set to match that of the actual producer. To consume a Logic Driven dynamic exchange, the Exchange ID must be in the range of 32769 - 49151. To consume a static exchange, the Exchange ID must be in the range of 1 - 16383.

It is the user's responsibility to prevent quick reuse of a Produced Exchange ID.

Effect of Resetting/Restarting the Ethernet Module

If the Ethernet Interface is restarted via either the push-button or a station manager restart command, the PLC CPU will set the exchange status words of logic driven exchanges associated with that particular Ethernet Interface to IPLC_FAILED_ADAPTER (18) and terminate all associated logic driven exchanges. The user program should monitor the LAN Interface Status Word and the Exchange Status Words of logic driven exchanges and re-establish exchanges when appropriate.

If there are multiple Ethernet Interface Modules in the system and only one of them is restarted, only the logic driven exchanges assigned to that Ethernet Module are terminated.

Exchange Status Word

The Exchange Status Word provides an ongoing status for the exchange. As with static exchanges, the Exchange Status Word will be updated throughout the life of the exchange. The format of the Exchange Status Word is shown below. No length field is provided since it is always assumed to

have a length of 1 WORD. The NULL selector cannot be used for the Memory Type since the Exchange Status Word must be present.

Table 12-18. Format of Exchange Status Word Address

Word	Description
1	**PLC Memory Type** Refer to Table 12-19 for possible memory types.
2	**PLC Memory Offset** Zero-based offset into memory type. Refer to Table 12-20 for offset calculations.
3	**Reserved** Must be set to 0.
4	**Reserved** Must be set to 0.

The following table shows the expected values that can be written to the Exchange Status Word for Logic Driven exchanges.

Table 12-19. Exchange Status Word Values

Status Mnemonic	Value	Production Exchanges	Consumption Exchanges
IPLC_NO_NEW_DATA	0	√	√
IPLC_NO_ERROR	1	√	
IPLC_NEW_DATA	1		√
GOOD_DATA_UNSYNC_TIMESTAMP	3	√	√
IPLC_SPEC_MISMATCH	4	√	√
IPLC_REFRESH_ERR	6	√	√
IPLC_DATA_AND_REFRESH_ERROR	7	√	√
IPLC_NO-NETWORK	10	√	√
IPLC_LOCAL_NO_RESOURCE	12	√	√
IPLC_WAITING_NAME_RESOLUTION	16	√	√
IPLC_FAILED_ADAPTER	18	√	√
IPLC_MOD_NOT_CAPABLE	22	√	
IPLC_CREATE_IN_PROGRESS	24		√
IPLC_CREATE_NO_RESPONSE	26	√	√
IPLC_CREATE_FAILED	28	√	√
IPLC_EXCHANGE_DELETED	30	√	√

Variables Address Definition

The following format is used for each of the variables in the variable list.

Table 12-20. Format of Variables

Word	Description
1	**PLC Memory Type** Refer to Table 12-23 for possible memory types.
2	**PLC Memory Offset** Zero-based offset into memory type. Refer to Table 12-23 for offset calculations.
3	**Reserved** Must be set to 0.
4	**Reserved** Must be set to 0.
5	**Variable Length (in bytes)** Max 1400 bytes. Refer to Table 12-23 for length calculations.
6	**Reserved** Must be set to 0.

Timestamp

The following format is used for the Timestamp Address. The user may specify the NULL selector (255) for the memory type field if they do not wish to have the timestamp available.

Table 12-21. Format of Timestamp Address

Word	Description
1	**PLC Memory Type** Refer to Table 12-23 for possible memory types.
2	**PLC Memory Offset** Zero-based offset into memory type. Refer to Table 12-23 for offset calculations.
3	**Reserved** Must be set to 0.
4	**Reserved** Must be set to 0.

The length of the timestamp is always 4 words as shown in the table below.

Table 12-22. POSIX Clock Timestamp Format

Word	Description
1	POSIX Clock – Seconds value (low word)
2	POSIX Clock – Seconds value (high word)
3	POSIX Clock – Nanoseconds value (low word)
4	POSIX Clock – Nanoseconds value (high word)

Memory Types for Exchange Status Word, Variables, and Timestamp

The following table should be used in constructing the addresses for exchange status words, variables, and timestamps.

Table 12-23. PLC Memory Type Formatting

PLC Memory Type	Code	Exchange Status Word	Time stamp	Produced Variable	Consumed Variable	Offset Calculation (bytes)	Length Calculation (bytes)
%R memory	8	√	√	√	√	$(Ref_{start}-1)*2$	$(Ref_{(end+1)} - Ref_{start})*2$
%AI memory	10	√	√	√	√		
%AQ memory	12	√	√	√	√		
%I memory	16	√	√	√	√	$(Ref_{start}-1)/8$ (Discard the remainder.)	$(Ref_{(end+8)} - Ref_{start})/8$ (Discard the remainder.)
%Q memory	18	√	√	√	√		
%T memory	20	√	√	√	√		
%M memory	22	√	√	√	√		
%SA memory	24			√	√		
%SB memory	26			√			
%SC memory	28	√	√	√	√		
%S memory	30			√			
%G memory	56			√			
%I override table	114			√			
%Q override table	116			√			
%T override table	118			√			
%M override table	120			√			
%SA override table	122			√			
%SB override table	124			√			
%SC override table	126			√			
S override table	1128			√			
%G override table	130			√			
NULL (no address)	255		√			0	0

PID

The Proportional plus Integral plus Derivative (PID) control function is the best known general purpose algorithm for closed loop process control. The Series 90 PID function block compares a Process Variable feedback with a desired process Set Point and updates a Control Variable output based on the error.

The block uses PID loop gains and other parameters stored in an array of 40 16 bit words (discussed on page 12-90) to solve the PID algorithm at the desired time interval. All parameters are 16 bit integer words for compatibility with 16 bit analog process variables. This allows %AI memory to be used for input Process Variables and %AQ to be used for output Control Variables. The example shown below includes typical inputs.

```
                                            _____
                                           |      |
                              %S00007       |      |
                 (enable)     —| |——     —|■ PID ■|—(ok)   Power flow out if OK
                                           |      |
                                           | IND  |
                                           |      |
            (set point)  %R00010       —|SP  CV|— %AQ0001   Control Variable
                          +21000           |      |         +25000
                                           |      |
     (process variable)  %AI0001       —|PV    |
                          +20950           |      |
                                           |      |
                          %M0001           |      |
                         —| |——         |MAN   |
                                           |      |
                                           |      |
                          %M0002           |      |
                         —| |——         |UP    |
                                           |      |
                          %M0002           |      |
                         —| |——         |DN    |
                                           |      |
                                           |_____|

                          %R00100            RefArray is 40 %R words

                    (reference array address)
```
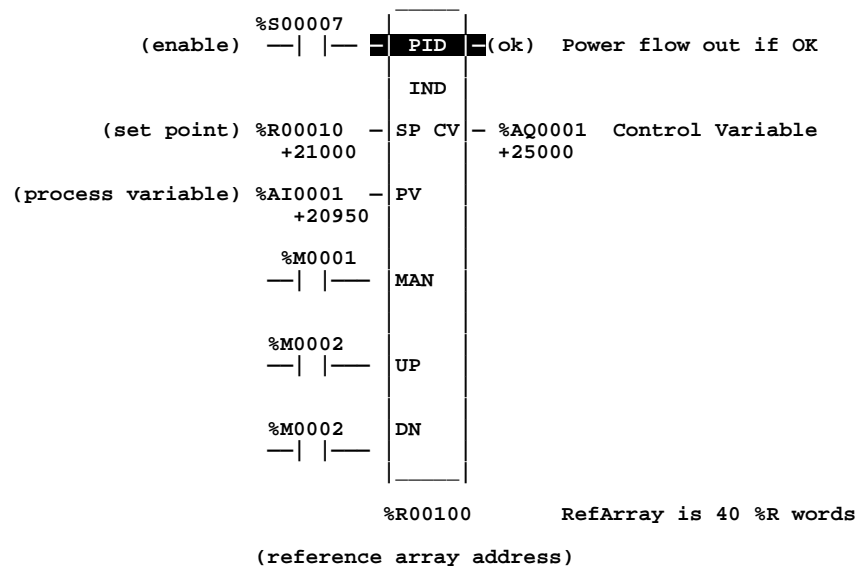
As the input Set Point and Process Variable and output Control Variable terms are used so frequently, they will be abbreviated as SP, PV and CV. As scaled 16 integer numbers, many parameters must be defined in either PV counts or units or CV counts or units. For example, the SP input must be scaled over the same range as PV as the PID block calculates the error by subtracting these two inputs. The PV and CV Counts may be –32000 or 0 to 32000 matching analog scaling or from 0 to 10000 to display variables as 0.00% to 100.00%. The PV and CV Counts do not have to have the same scaling, in which case there will be scale factors included in the PID gains.

Note

The PID will not execute more often than once every 10 milliseconds. This could change your results if you set it up to execute every sweep and the sweep is under 10 milliseconds. In such a case, the PID function will not run until enough sweeps have occurred to accumulate an elapsed time of 10 milliseconds; for example, if the sweep time is 9 milliseconds, the PID function will execute every other sweep with an elapsed time of 18 milliseconds for every time it executes.

Parameters:

Parameter	Description
enable	When enabled through a contact, the PID function is performed.
SP	SP is the control loop or process set point. Set using PV Counts, the PID adjusts the output CV so that PV matches SP (zero error).
PV	Process Variable input from the process being controlled, often a %AI input.
MAN	When energized to 1 (through a contact), the PID block is in **MANUAL** mode. If the PID block is on manual off, the PID block is in automatic mode.
UP	If energized along with MAN, it adjusts the CV up by 1 CV per solution.*
DN	If energized along with MAN, it adjusts the CV down by 1 CV per solution.*
RefArray Address	Address is the location of the PID control block information (user and internal parameters). Uses 40 %R words that cannot be shared.
ok	The ok output is energized when the function is performed without error. It is off if error(s) exist.
CV	CV is the control variable output to the process, often a %AQ analog output.

*Incremented (UP parameter) or decremented (DN parameter) by one (1) per access of the PID function.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
SP		•	•	•	•		•	•	•	•	•	
PV		•	•	•	•		•	•	•	•		
MAN	•											
UP	•											
DN	•											
address								•				
ok	•											•
CV		•	•	•	•		•	•	•	•		

• Valid reference or place where power may flow through the function.

PID Parameter Block:

Besides the 2 input words and the 3 Manual control contacts, the PID block uses 13 of the parameters in the RefArray. These parameters must be set before calling the block. The other parameters are used by the PLC and are non-configurable. The %Ref shown in the table below is the same RefArray Address at the bottom of the PID block. The number after the plus sign is the offset in the array. For example, if the RefArray starts at %R100, the %R113 will contain the Manual Command used to set the Control Variable and the integrator in Manual mode.

Table 12-4. PID Parameters Overview

Register	Parameter	Low Bit Units	Range of Values
%Ref+0000	Loop Number	Integer	0 to 255 (for user display only)
%Ref+0001	Algorithm	N/A; set and maintained by the PLC	Non-configurable
%Ref+0002	Sample Period	10 milliseconds	0 (every sweep) to 65535 (10.9 Min). Use at least 10 for 90-30 PLCs (see Note on page 12-88).
%Ref+0003	Dead Band +	PV Counts	0 to 32000 (never negative)
%Ref+0004	Dead Band —	PV Counts	–32000 to 0 (never positive)
%Ref+0005	Proportional Gain –Kp	0.01 CV%/PV%	0 to 327.67 %/%
%Ref+0006	Derivative Gain–Kd	0.01 seconds	0 to 327.67 sec
%Ref+0007	Integral Rate–Ki	Repeat/1000 Sec	0 to 32.767 repeat/sec
%Ref+0008	CV Bias/Output Offset	CV Counts	–32000 to 32000 (add to integrator output)
%Ref+0009	Upper Clamp	CV Counts	–32000 to 32000(>%Ref+10) output limit
%Ref+0010	Lower Clamp	CV Counts	–32000 to 32000(<%Ref+09) output limit
%Ref+0011	Minimum Slew Time	Second/Full Travel	0 (none) to 32000 sec to move 32000 CV
%Ref+0012	Config Word	Low 5 bits used	Bit 0 to 2 for Error+/–, OutPolarity, Deriv.
%Ref+0013	Manual Command	CV Counts	Tracks CV in Auto or Sets CV in Manual
%Ref+0014	Control Word	Maintained by the PLC, *unless* Bit 1 is set.	PLC maintained unless set otherwise: low bit sets Override if 1 (see description in the "PID Parameters Details" table on page 12-91)
%Ref+0015	Internal SP	N/A; set and maintained by the PLC	Non-configurable
%Ref+0016	Internal CV	N/A; set and maintained by the PLC	Non-configurable
%Ref+0017	Internal PV	N/A; set and maintained by the PLC	Non-configurable
%Ref+0018	Output	N/A; set and maintained by the PLC	Non-configurable

Table 12-4. PID Parameters Overview - Continued

Register	Parameter	Low Bit Units	Range of Values
%Ref+0019	Diff Term Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0020 and %Ref+0021	Int Term Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0022	Slew Term Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0023	Clock	N/A; set and	
%Ref+0024		maintained by	Non-configurable
%Ref+0025	(time last executed)	the PLC	
%Ref+0026	Y Remainder Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0027	Lower Range for SP, PV	PV Counts	–32000 to 32000 (>%Ref+28) for display
%Ref+0028	Upper Range for SP, PV	PV Counts	–32000 to 32000 (<%Ref+27) for display
%Ref+0029 • %Ref+0034	Reserved for internal use	N/A	Non-configurable
%Ref+0035 • %Ref+0039	Reserved for external use	N/A	Non-configurable

Note that every PID block call must use a different 40-word array even if all 13 user parameters are the same because other words in the array are used for internal PID data storage. Make sure the array does not extend beyond the end of memory.

To configure the user parameters, select the PID function and press **F10** to zoom in to a screen displaying User Parameters; then use arrow keys to select fields and type in desired values. You can use 0 for most default values, except the CV Upper Clamp, which must be greater than the CV Lower Clamp for the PID block to operate. Note that the PID block does **not** pass power if there is an error in User Parameters, so monitor with a temporary coil while modifying data.

Once suitable PID values have been chosen, they should be defined as constants in the BLKMOV so that they can be used to reload default PID user parameters if needed.

Operation of the PID Instruction

Normal Automatic operation is to call the PID block every sweep with power flow to Enable and no power flow to Manual input contacts. The block compares the current PLC elapsed time clock with the last PID solution time stored in the internal RefArray. If the time difference is greater than the sample period defined in the third word (%Ref+2) of the RefArray, the PID algorithm is solved using the time difference and both the last solution time and Control Variable output are updated. In Automatic mode, the output Control Variable is placed in the Manual Command parameter %Ref+13.

If power flow is provided to both Enable and Manual input contacts, the PID block is placed in Manual mode and the output Control Variable is set from the Manual Command parameter %Ref+13. If either the UP or DN inputs have power flow, the Manual Command word is incremented or decremented by one CV count every PID solution. For faster manual changes of the output Control Variable, it is also possible to add or subtract any CV count value directly to/from the Manual Command word.

The PID block uses the CV Upper and CV Lower Clamp parameters to limit the CV output. If a positive Minimum Slew Time is defined, it is used to limit the rate of change of the CV output. If either the CV amplitude or rate limit is exceeded, the value stored in the integrator is adjusted so that CV is at the limit. This anti-reset windup feature (defined on page 12-95) means that even if the error tried to drive CV above (or below) the clamps for a long period of time, the CV output will move off the clamp as soon as the error term changes sign.

This operation, with the Manual Command tracking CV in Automatic mode and setting CV in Manual mode, provides a bumpless transfer between Automatic and Manual modes. The CV Upper and Lower Clamps and the Minimum Slew Time still apply to the CV output in Manual mode and the internal value stored in the integrator is updated. This means that if you were to step the Manual Command in Manual mode, the CV output will not change any faster that the Minimum Slew Time (Inverse) rate limit and will not go above or below the CV Upper or CV Lower Clamp limits.

Note

A specific PID function should not be called more than once per sweep.

The following table provides more details about the parameters discussed briefly in Table 12-4. The number in parentheses after each parameter name is the offset in the RefArray.

Table 12-5. PID Parameters Details

Data Item	Description
Loop Number (00)	This is an optional parameter available to identify a PID block. It is an unsigned integer that provides a common identification in the PLC with the loop number defined by an operator interface device. The loop number is displayed under the block address when logic is monitored from the Logicmaster 90-70 software.
Algorithm (01)	An unsigned integer that is set by the PLC to identify what algorithm is being used by the function block. The ISA algorithm is defined as algorithm 1, and the independent algorithm is identified as algorithm 2.
Sample Period (02)	The shortest time, in 10 millisecond increments, between solutions of the PID algorithm. For example, use a 10 for a 100 millisecond sample period. The UINT value can be up to 65535 for a sample period of 10.9 minutes. If it is 0, the algorithm is solved every time the block is called (see section below on PID block scheduling). The PID algorithm is solved only if the current PLC elapsed time clock is at or later than the last PID solution time plus this Sample Period. Remember, that the 90-70 will not use a solution time less than 10 milliseconds (see Note on page 12-88); so sweeps will be skipped for smaller sweep times. This function compensates for the actual time elapsed since the last execution, within 100 microseconds. If this value is set to 0, the function is executed each time it is enabled; however, it is restricted to a minimum of 10 milliseconds as noted above.
Dead Band (+/—) (03/04)	INT values defining the upper (+) and lower (–) Dead Band limits in PV Counts. 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 be greater than 0 and the (–) value less than 0 or the PID block will not function. *You should leave these at 0 until the PID loop gains are setup or tuned.* After that, you may want to add Dead Band to avoid small CV output changes due to small variations in error, perhaps to reduce mechanical wear.
Proportional Gain–Kp (05)	This INT number, called the Controller gain, Kc, in the ISA version, determines the change in CV in CV Counts for a 100 PV Count change in the Error term. It is displayed as 0.00 %/% with an implied decimal point of 2 . For example, a Kp entered as 450 will be displayed as 4.50 and will result in a Kp*Error/100 or 450*Error/100 contribution to the PID Output. Kp is generally the first gain set when adjusting a PID loop.
Derivative Gain–Kd (06)	This INT number determines the change in CV in CV Counts if the Error or PV changes 1 PV Count every 10 milliseconds. Entered as a time with the low bit indicating 10 milliseconds, it is displayed as 0.00 Seconds with an implied decimal point of 2. For example, a Kd entered as 120 will be displayed as 1.20 Sec and will result in a Kd * delta Error/delta time or 120*4/3 contribution to the PID Output if Error was changing by 4 PV Counts every 30 milliseconds. Kd can be used to speed up a slow loop response, but is very sensitive to PV input noise.
Integral Rate Gain–Ki (07)	This INT number determines the change in CV in CV Counts if the Error were a constant 1 PV Count. It is displayed as 0.000 Repeats/Sec with an implied decimal point of 3. For example, a Ki entered as 1400 will be displayed as 1.400 Repeats/Sec and will result in a Ki * Error *dt or 1400 * 20 * 50/1000 contribution to PID Output for an Error of 20 PV Counts and a 50 millisecond PLC sweep time (Sample Period of 0). Ki is usually the second gain set after Kp.
CV Bias/Output Offset (08)	An INT value in CV Counts added to the PID Output before the rate and amplitude clamps. It can be used to set non-zero CV values if only Kp Proportional gains are used, or for feed forward control of this PID loop output from another control loop.

Table 12-5. PID Parameters Details - Continued

Data Item	Description
CV Upper and Lower Clamps (09/10)	INT values in CV Counts that define the highest and lowest value for CV. These values are required and 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 for the LM90 or ADS PID display. The block has anti-reset windup to modify the integrator value when a CV clamp is reached.
Minimum Slew Time (11)	A positive UINT value to define the 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 be changed. If positive, CV can not change more than 32000 CV Counts times Delta Time (seconds) divided by Minimum Slew Time. For example, if the Sample Period was 2.5 seconds and the Minimum Slew Time is 500 seconds, CV can not change more than 32000*2.5/500 or 160 CV Counts per PID solution. As with the CV Clamps, there is an anti-windup feature that adjusts the integrator value if the CV rate limit is exceeded. If Minimum Slew Time is 0, there is no CV rate limit. Make sure you set Minimum Slew Time to 0 while you are tuning or adjusting PID loop gains.
Config Word	The low 5 bits of this word are used to modify three standard PID settings. The other bits should be set to 0. Set the low bit to 1 to modify the standard PID Error Term from the normal (SP – PV) to (PV – SP), reversing the sign of the feedback term. This is for Reverse Acting controls where the CV must go down when the PV goes up. Set the second bit to a 1 to invert the Output Polarity so that CV is the negative of the PID output rather than the normal positive value. Set the fourth bit to 1 to modify the Derivative Action from using the normal change in the Error term to the change in the PV feedback term. The low 5 bits in the Config Word are defined in detail below: **Bit 0** = Error Term. When this bit is set to 0, the error term is SP — PV. When this bit is set to 1, the error term is PV — SP. **Bit 1** = Output Polarity. When this bit is set to 0, the CV output represents the output of the PID calculation. When it is set to 1, the CV output represents the negative of the output of the PID calculation. **Bit 2** = Derivative action on PV. When this bit is set to 0, the derivative action is applied to the error term. When it is set to 1, the derivative action is applied to PV. All remaining bits should be zero. **Bit 3** = Deadband action. When the Deadband action bit is set to zero, then no deadband action is chosen. If the error is within the deadband limits, then the error is forced to be zero. Otherwise the error is not affected by the deadband limits. If the Deadband action bit is set to one, then deadband action is chosen. If the error is within the deadband limits, then the error is forced to be zero. If, however, the error is outside the deadband limits, then the error is reduced by the deadband limit (error = error – deadband limit). **Bit 4** =Anti-reset windup action. When this bit is set to zero, the anti-reset windup action uses a reset back calculation. When the output is clamped, this replaces the accumulated Y remainder value (defined on page 12-95) with whatever value is necessary to produce the clamped output exactly. When the bit is set to one, this replaces the accumulated Y term with the value of the Y term at the start of the calculation. In this way, the pre-clamp Y value is held as long as the output is clamped. **NOTE:** The anti-reset windup action bit is only available on release 6.01 or later 90-70 CPUs. Remember that the bits are set in powers of 2. For example, to set Config Word to 0 for default PID configuration, you would add 1 to change the Error Term from SP–PV to PV–SP, or add 2 to change the Output Polarity from CV = PID Output to CV = – PID Output, or add 4 to change Derivative Action from Error rate of change to PV rate of change, etc.

Table 12-5. PID Parameters Details - Continued

Data Item	Description
Manual Command (13)	This is an INT value 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 integrator within the Upper and Lower Clamp and Slew Time limits.
Control Word (14)	This is an internal parameter that is normally left at 0. If the Override low bit is set to 1, this word and other internal SP, PV and CV parameters must be used for remote operation of this PID block (see below). This allows remote operator interface devices, such as a computer, to take control away from the PLC program. Caution: if you do not want this to happen, 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. A discrete data structure with the first five bit positions in the following format: **Bit: Word Value: Function: Status or External Action if Override bit set to 1:** 0 1 Override If 0, monitor block contacts below. If 1, set them externally. 1 2 Manual/ Auto If 1, block is in **Manual** mode; other numbers it is in **Automatic** mode. 2 4 Enable Should normally be 1; otherwise block is never called. 3 8 UP/Raise If 1 and Manual (Bit 1) is 1, CV is being incremented every solution. 4 16 DN/Lower If 1 and Manual (Bit 1) is 1, CV is being incremented every solution.
SP (15)	(Non-configurable–set and maintained by the PLC) Tracks SP in; must be set externally if Override = 1.
CV (16)	(Non-configurable–set and maintained by the PLC) Tracks CV out.
PV (17)	(Non-configurable–set and maintained by the PLC) Tracks PV in; must be set externally if Override bit = 1.
Output (18)	(Non-configurable–set and maintained by the PLC) This is a signed word value representing the output of the function block before the application of the optional inversion. If no output inversion is configured and the output polarity bit in the control word is set to 0, this value will equal the CV output. If inversion is selected and the output polarity bit is set to 1, this value will equal the negative of the CV output.
Diff Term Storage (19)	Used internally for storage of intermediate values. *Do not write to this location.*
Int Term Storage (20/21)	Used internally for storage of intermediate values. *Do not write to this location.*
Slew Term Storage (22)	Used internally for storage of intermediate values. *Do not write to this location.*
Clock (23–25)	Internal elapsed time storage (time last PID executed). *Do not write to these locations.*
Y Remainder (26)	Holds remainder for integrator division scaling for 0 steady state error.
Lower and Upper Range (27/28)	Optional INT values in PV Counts that define the highest and lowest display value for the SP and PV Logicmaster **Zoom** key horizontal bar graph and ADS PID faceplate display.
Reserved (29–34 and 35–39)	29–34 are reserved for internal use; 35–39 are reserved for external use. They are reserved for GE Fanuc use, and cannot be used for other purposes.

Internal Parameters in RefArray

As described in Table 12-6 on the previous pages, the PID block reads 13 user parameters and uses the rest of the 40 word RefArray for internal PID storage. Normally you would not need to change any of these values. If you are calling the PID block in Auto mode after a long delay, you may want to use SVC_REQ #16 to load the current PLC elapsed time clock into %Ref+23 to update the last PID solution time to avoid a step change on the integrator. If you have set the Override low bit of the Control Word (%Ref+14) to 1, the next four bits of the Control Word must be set to control the PID block input contacts (as described in Table 12-5 on the previous pages), and the Internal SP and PV must be set as you have taken control of the PID block away from the ladder logic. The internal parameter words are:

PID Algorithm Selection (PIDISA or PIDIND) and Gains

The PID block can be programmed selecting either the Independent (PID_IND) term or standard ISA (PID_ISA) versions of the PID algorithm. The only difference in the algorithms is how the Integral and Derivative gains are defined. To understand the difference, you need to understand the following:

Both PID types calculate the Error term as SP – PV, which can be changed to Reverse Acting mode PV – SP if the Error Term (low bit 0 in the Config Word %Ref+12) is set to 1. Reverse Acting mode may be used if you want the CV output to move in the opposite direction from PV input changes (CV down for PV up) rather than the normal CV up for PV up.

Error = (SP – PV)　　　　　or (PV – SP) if low bit of Config Word set to 1

The Derivative is normally based on the change of the Error term since the last PID solution, which may cause a large change in the output if the SP value is changed. If this is not desired, the third bit of the Config Word can be set to 1 to calculate the Derivative based on the change of the PV. The dt (or Delta Time) is determined by subtracting the last PID solution clock time for this block from the current PLC elapsed time clock.

dt = Current PLC Elapsed Time clock – PLC Elapsed Time Clock at Last PID solution

Derivative = (Error – previous Error)/dt　　　or　(PV – previous PV)/dt if 3rd bit of Config Word set to 1

The Independent term PID (PID_IND) algorithm calculates the output as:

PID Output = Kp * Error + Ki * Error * dt + Kd * Derivative + CV Bias

The standard ISA (PID_ISA) algorithm has a different form:

PID Output = Kc * (Error + Error * dt/Ti + Td * Derivative) + CV Bias

where Kc is the controller gain, and Ti is the Integral time and Td is the Derivative time. The advantage of ISA is that adjusting the Kc changes the contribution for the integral and derivative terms as well as the proportional one, which may make loop tuning easier. If you have PID gains in terms or Ti and Td, use

Kp = Kc　　　　　Ki = Kc/Ti　　　and　　　　　　　Kd = Kc/Td

to convert them to use as PID User Parameter inputs.

The CV Bias term above is an additive term separate from the PID components. It may be required if you are using only Proportional Kp gain 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 an Integral Ki gain is used, the CV Bias would normally be 0 as the integrator acts as an automatic bias. Just start up in Manual mode and use the Manual Command word (%Ref+13) to set the integrator to the desired CV, then switch to Automatic mode. This also works if Ki is 0, except the integrator will not be adjusted based on the Error after going into Automatic mode.

The following diagram shows how the PID algorithms work:
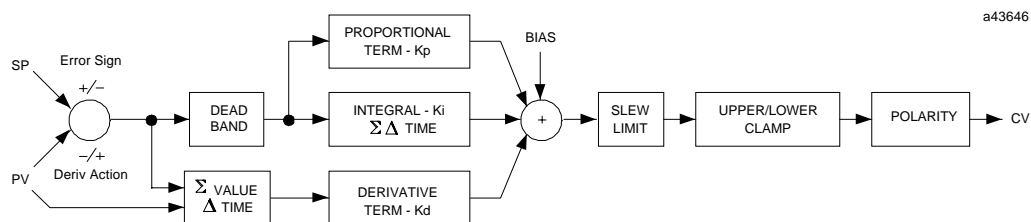
Figure 12-1. Independent Term Algorithm (PIDIND)

The ISA Algorithm (PIDISA) is similar except the Kp gain is factored out of Ki and Kd so that the integral gain is Kp * Ki and derivative gain is Kp * Kd. The Error sign, DerivAction and Polarity are set by bits in the Config Word user parameter.

CV Amplitude and Rate Limits

The block does not send the calculated PID Output directly to CV. Both PID algorithms can impose amplitude and rate of change limits on the output Control Variable. The maximum rate of change is determined by dividing the maximum 100% CV value (32000) by the Minimum Slew Time, if specified as greater than 0. For example, if the Minimum Slew Time is 100 seconds, the rate limit will be 320 CV counts per second. If the dt solution time was 50 milliseconds, the new CV output can not 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 and CV Lower Clamp values. If either limit is exceeded, the CV output is set to the clamped value. If either rate or amplitude limits are exceeded modifying CV, the internal integrator value is adjusted to match the limited value to avoid reset windup.

Finally, the block checks the Output Polarity (2nd bit of the Config Word %Ref+12) and changes the sign of the output if the bit is 1.

CV = Clamped PID Output   or – Clamped PID Output if Output Polarity bit set

If the block is in Automatic mode, the final CV is placed in the Manual Command %Ref+13. If the block is in Manual mode, the PID equation is skipped as CV is set by the Manual Command, but all the rate and amplitude limits are still checked. That means that the Manual Command can not change the output above the CV Upper Clamp or below the CV Lower Clamps and the output can not change faster than the Minimum Slew Time allowed.

Sample Period and PID Block Scheduling

The PID 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, possibly with a pure time delay. The PID 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 does the PV respond when the CV is changed. As discussed in the Setting Loop Gains section below, the total time constant, Tp+Tc, for a first order system is the time required for PV to reach 63% of its final value when CV is stepped. The PID 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 integrator will round down to 0. For example, a very slow process that takes 10 hours or 36000 seconds to reach the 63% level should have a Sample Period of 40 seconds or longer.

Unless the process is very fast, it is not usually necessary to use a Sample Period of 0 to solve the PID algorithm every PID sweep. If many PID loops are used with a Sample Period greater than the sweep time, there may be wide variations in PLC sweep time if many loops end up solving the algorithm at the same time. The simple solution is to sequence a one or more 1 bits through an array of bits set to 0 that is being used to enable power flow to individual PID blocks.

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 we change CV by a fixed amount, or what is the open loop gain?

2. How fast does the system respond, or how quick 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:

PV(s)/CV(s) = G(s) = K * e **(–Tp s)/(1 + Tc s)

Plotting a step response at time t0 in the time domain provides an open loop unit reaction curve:
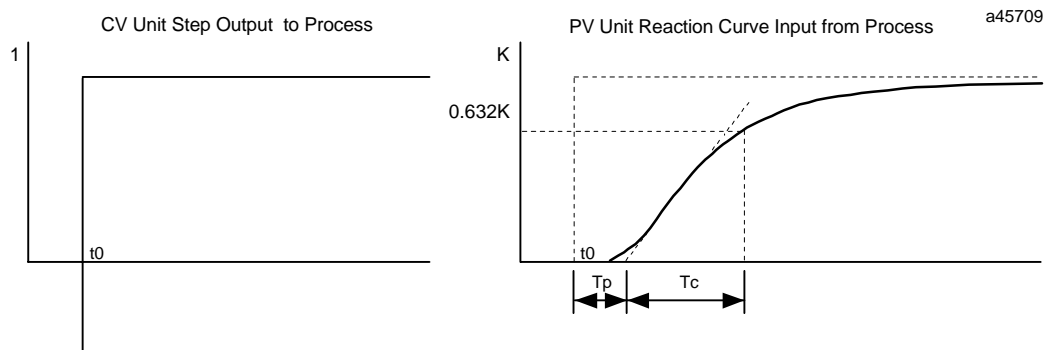
The following process model parameters can be determined from the PV unit reaction curve:

K	Process open loop gain = final change in PV/change in CV at time t0 (Note no subscript on K)
Tp	Process or pipeline time delay or dead time after t0 before the process output PV starts moving
Tc	First order Process time constant, time required after Tp for PV to reach 63.2% of the final PV

Usually the quickest way to measure these parameters is by putting the PID block in Manual mode and making a small step in CV output, by changing the Manual Command %Ref+13, and 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 size may be from 2 to 10% of the difference between the CV Upper and CV Lower Clamp values .

Setting User Parameters Including Tuning Loop Gains

As all PID parameters are totally dependent on the process being controlled, there are no predetermined values that will work, however, it is usually a simple, iterative procedure to find acceptable loop gain.

1. Set all the User Parameters to 0, then set the CV Upper and CV Lower Clamps to the highest and lowest CV expected. Set the Sample Period to the estimated process time constant (above)/10 to 100.

2. Put block in Manual mode and set Manual Command (%Ref+13) at different values to check if CV can be moved to Upper and Lower Clamp. Record 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 2 to 3 cycles. This may required reducing Kp. Also try different 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 Config Word Error Sign or Polarity bits

Setting Loop Gains — Ziegler and Nichols Tuning Approach

Once the three process model parameters, K, Tp and Tc, are determined, they can be used to estimate initial PID loop gains. The following approach, developed by Ziegler and Nichols in the 1940's, is designed to provide good response to system disturbances with gains producing a 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. Calculate the Reaction rate:

   $R = K/Tc$

2. For Proportional control only, calculate Kp as

   $Kp = 1/(R * Tp) = Tc/(K * Tp)$

3. For Proportional and Integral control, use

   $Kp = 0.9/(R * Tp) = 0.9 * Tc/(K * Tp)$

   $Ki = 0.3 * Kp/Tp$

4. For Proportional, Integral and Derivative control, use

   $Kp = G/(R * Tp)$ where G is from 1.2 to 2.0

   $Ki = 0.5 * Kp/Tp$

   $Kd = 0.5 * Kp * Tp$

5. Check that the Sample Period is in the range $(Tp + Tc)/10$ to $(Tp + Tc)/1000$

Another approach, the "Ideal Tuning" procedure, is designed to provide the best response to SP changes, delayed only by the Tp process delay or dead time.

   $Kp = 2 * Tc/(3 * K * Tp)$

   $Ki = Tc$

   $Kd = Ki/4$ if Derivative term is used

Once initial gains are determined, they must be converted to integer User Parameters. To avoid scaling problems, the Process gain, K, should be calculated as a change in input PV Counts divided by the output step change in CV Counts and not in process PV or CV engineering units. All times should also be specified in seconds. Once Kp, Ki and Kd are determined, Kp and Kd can be multiplied by 100 and entered as integer while Ki can be multiplied by 1000 and entered into the User Parameter %RefArray.

Sample PID Call

The following example has a Sample Period of 100 millisecond, a Kp gain of 4.00 and a Ki gain of 1.500. The Set Point is stored in %R1 with the Control Variable output in %AQ2 and the Process Variable returned in %AI3. CV Upper and CV Lower Clamps must be set, in this case to 20000 and 400, and an optional small Dead Band of +5 and –5 has been included. The 40 word RefArray starts in %R100. Normally User Parameters are set in the RefArray with the PID Zoom key **F10**, but %M6 can be set to reinitialize the 14 words starting at %R102 (%Ref+2) from constants stored in logic.

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 and integrator by 1 every 100 millisecond solution. For faster manual operation, bits %M2 and %M3 can be used to add or subtract the value in %R2 to/from %R113 every PLC sweep. The %T1 output is on when the PID is OK.

```
 %M0006        _____            _____                        _____
 —| |—        | BLK_ |         | BLKMV |——————————————————  | BLKMV |  —
               | CLR_ |         | INT   |                    | INT   |
               | WORD |         |       |                    |       |
 %R00100—      | IN   | CONST — | IN1 Q |—%R00102   CONST — | IN1 Q |— %R00109
               | LEN  | +00010  |       |           +20000  |       |
               | 00035|         |       |                   |       |
               |_____| CONST — | IN2   |           CONST — | IN2   |
                        +00005  |       |           +00400  |       |
                                |       |                   |       |
                        CONST — | IN3   |           CONST — | IN3   |
                        +00005  |       |           +00000  |       |
                                |       |                   |       |
                        CONST — | IN4   |           CONST — | IN4   |
                        +00400  |       |           +00000  |       |
                                |       |                   |       |
                        CONST — | IN5   |           CONST — | IN5   |
                        +00000  |       |           +00000  |       |
                                |       |                   |       |
                        CONST — | IN6   |           CONST — | IN6   |
                        +01500  |       |           +00000  |       |
                                |       |                   |       |
                        CONST — | IN7   |           CONST — | IN7   |
                        +00000  |_____|             +00000  |_____|


 ALW_ON                                                      _____      %T0001
 —| |———————————————————————————————————————————————————   | PID_ |—( )—
                                                            | IND  |
                                                  %R0001—   | SP CV|— %AQ002
                                                            |      |
                                                  %AI0003—  | PV   |
                                                            |      |
 %M0001                                                     |      |
 —| |———————————————————————————————————————————————————   | MAN  |
                                                            |      |
                                        %M0004              |      |
                                        —| |—————           | UP   |
                                                            |      |
                                        %M0005              |      |
                                        —| |—————           | DN   |
                                                            |_____|
                                                   %R00100


 %M0002         _____
 —| |————      | ADD_ |————
               | INT  |
 %R00113—      | I1  Q|— %R00113
               |      |
 %R00002—      | I2   |
               |      |
               |_____|


 %M0003         _____
 —| |————      | SUB_ |—
               | INT  |
 %R00113—      | I1  Q|— %R00113
               |      |
 %R00002—      | I2   |
               |      |
               |_____|
```

CPU Performance Data

This appendix contains instruction and overhead timing for each Series 90-70 CPU module. This timing information can be used to predict CPU sweep times.

Instruction Timing

The Series 90-70 PLC supports many different functions and function blocks. Table A-1, beginning on the next page, lists the execution time in microseconds and the memory size in bytes for each function supported by these CPUs:

1. Model CPM 924/925

2. Model CPM 914/915

3. Model CPU 781/782

4. Model CPU 788/89

5. Models CPU 731R and later, CPU 732, CPU 771P and later, and CPU 772

6. Models CPX 935 and CGR 935

7. Model CPX 928

8. Models CPX 772 and CGR 772, and CPX 782

Note

All timing information for all CPUs was not available at the time this manual was printed. Additional information will be added to a future edition.

Two execution times are shown for each function:

Execution Time	Description
Enabled	Time required to execute the function or function block when power flows into and out of the function. Typically, best-case times are when the data used by the block is contained in user RAM (word-oriented memory).
Disabled	Time required to execute the function when it is not enabled.

Note

Timers are updated each time they are encountered in the logic by the amount of time consumed by the last sweep.

Table A-1. Instruction Timing

Function Group	Function	Enabled				Disabled				Increment				Size
		924/ 925	914/ 915	781/782 788/789	731/732 771/772	924/ 925	914/ 915	781/782 788/789	731/732 771/772	924/ 925	914/ 915	781/782 788/789	731/732 771/772	
Timers	ONDTR	6.5	10.5	34.5	78	5.5	8	22.5	53	-	-	-	-	18
	OFDT	6	9.5	32.5	68	6	9.5	30	65.5	-	-	-	-	15
	TMR	7.5	10	37	79	8.5	10.5	32	62	-	-	-	-	15
Counters	UPCTR	8.5	11	28.5	67	8	11	29	63	-	-	-	-	18
	DNCTR	6	9.5	28.5	66	6	9.5	28	62.5	-	-	-	-	18
Math	ADD (INT)	5	6	22	34.5	4	5.5	14.5	18	-	-	-	-	15
	ADD (DINT)	7	9.5	34.5	63.5	6	6.5	19.5	30.5	-	-	-	-	15
	SUB (INT)	5	6.5	17	35	4	4.5	10.5	19	-	-	-	-	15
	SUB (DINT)	8	10	36	63.5	6	7.5	20	30	-	-	-	-	15
	MUL (INT)	5	6.5	21	38	4	4.5	11.5	18.5	-	-	-	-	15
	MUL (DINT)	8	10.5	35.5	76.5	5.5	6.5	19.5	31	-	-	-	-	15
	DIV (INT)	5.5	7	22.5	41	4.5	4.5	14.5	19	-	-	-	-	15
	DIV (DINT)	8.5	12	41	82	5.5	6.5	18	30.5	-	-	-	-	15
	MOD (INT)	5.5	7.5	23.5	41	4	5	12.5	18.5	-	-	-	-	15
	MOD (DINT)	10.5	14	46.5	85	5.5	7.5	21	31	-	-	-	-	15
	ABS (INT)	5.5	7.5	25.5	50	4.5	5.5	18	25.5	-	-	-	-	12
	ABS (DINT)	7	8.5	28.5	51.5	5	7	18.5	26.5	-	-	-	-	12
	SQRT (INT)	10	14	40.5	82.5	4	4	11	16.5	-	-	-	-	12
	SQRT (DINT)	15.5	27.5	80.5	166	5	6.5	16.5	26.5	-	-	-	-	12
Relational	EQ (INT)	6	8	21	37	3.5	4.5	12.5	17.5	-	-	-	-	15
	EQ (DINT)	6.5	9	30	58.5	5	7	18.5	30.5	-	-	-	-	15
	NE (INT)	6	8	20	36.5	3.5	4.5	14	18.5	-	-	-	-	15
	NE (DINT)	6.5	10.5	31	58	5	7	20	30.5	-	-	-	-	15
	GT (INT)	6.5	9	24	43.5	4	4.5	13.5	17.5	-	-	-	-	15
	GT (DINT)	7.5	10.5	28.5	61	5	7.5	18	30.5	-	-	-	-	15
	GE (INT)	6.5	9	25	41.5	3.5	4.5	14	17.5	-	-	-	-	15
	GE (DINT)	6.5	9.5	31.5	58.5	5	7	20	30.5	-	-	-	-	15
	LT (INT)	7.5	9.5	27.5	43.5	3.5	4.5	14.5	17.5	-	-	-	-	15
	LT (DINT)	7.5	10.5	35	61	5	7	20	30.5	-	-	-	-	15
	LE (INT)	6	8.5	20	41.5	3.5	4.5	14	17.5	-	-	-	-	15
	LE (DINT)	6.5	10.5	32	58.5	5	7	21.5	30.5	-	-	-	-	15
	CMP (INT)	10.5	12.5	34.5	62	9	14	33.5	55.5	-	-	-	-	21
	CMP (DINT)	12	14.5	41	79.5	9	12	32	56	-	-	-	-	21
	RANGE (INT)	10.5	16.5	46	82.5	5.5	8	26	33	-	-	-	-	
	RANGE (DINT)	9	14.5	53.5	96.5	8	9	21.5	34	-	-	-	-	
Bit Operation	AND (WORD)	9.5	13.5	53.5	116.5	5	7	28.5	35	0.7	1.5	8.2	18.1	18
	AND(DWORD)	9.5	15	58	123.5	6	7.5	25.5	36	1	2	10.1	24.4	18
	OR (WORD)	8.5	14	57	118	5.5	7	25	35	0.8	1.6	9	19.6	18
	OR (DWORD)	9.5	15	63	126.5	6.5	8.5	24.5	36	1.2	2.3	11.7	27.5	18
	XOR (WORD)	8.5	14	58.5	118	5	7	24.5	35	0.8	1.6	8.9	19.3	18
	XOR (DWORD)	9.5	15	65	124.5	5.5	7.5	26.5	34.5	1.1	2.3	11.6	26.6	18
	NOT (WORD)	7	10.5	42.5	76.5	5.5	6.5	22	32.5	0.3	0.7	3.5	8.4	15
	NOT (DWORD)	7.5	11	41	77.5	5.5	7	25.5	32	0.3	0.7	3.6	10.3	15
	MCMP (WORD)	14	24.5	97.5	213	7	10.5	36.5	56.5	0.6	1.3	3.8	8.4	30
	MCMP (DWORD)	15.5	27	100	221	7.5	11	35	56	1.3	2.7	7.7	16.9	30
	SHL (WORD)	11	17.5	61	137.5	5.5	8.5	25.5	41.5	0.6	1	3.9	10.2	24
	SHL (DWORD)	12	18.5	66.5	147	6	8.5	27	43	1.2	2.1	7.9	20.5	24
	SHR (WORD)	10.5	16.5	60.5	137.5	5	8	26.5	41	0.6	1	3.9	10.1	24
	SHR (DWORD)	12	18	68	147	6	8.5	25	43	1.2	2.1	7.9	20.3	24
	ROL (WORD)	12.5	18	55	117.5	5	7.5	24	33.5	0.6	1.1	4	9.4	18
	ROL (DWORD)	14	20	57.5	127	6	7.5	23.5	35.5	1.3	2.2	8.2	19	18
	ROR (WORD)	8.5	14	51	113	5	7	20.5	33.5	0.6	1.1	4.1	9.4	18
	ROR (DWORD)	10	16	55.5	122	6.5	7.5	21.5	35.5	1.3	2.3	8.2	19	18
	BTST (WORD)	7.5	11	46.5	92	5	7	22.5	35	-	-	-	-	18
	BTST (DWORD)	7	11	43	91	4	6	21.5	34	-	-	-	-	18
	BSET (WORD)	9	11	42.5	89.5	4	5.5	23	32.5	-	-	-	-	15
	BSET (DWORD)	7.5	10.5	40.5	88.5	4	5.5	18.5	31.5	-	-	-	-	15
	BCLR (WORD)	6	10.5	44	89.5	4	5.5	20.5	33.5		-	-	-	15
	BCLR (DWORD)	6	10	42	88.5	4	5.5	20	32.5	-				15
	MOVE (BIT)	14	19	58	117.5	3.5	5.5	21	31	0.8	1.2	3.2	6	15
	BPOS (WORD)	10	15	53	108	4	6	22.5	34	0.2	0.4	1.8	4.1	18
	BPOS (DWORD)	13	20.5	81	157.5	4	6	22	35	0.4	0.8	3.1	6.9	18
	SHFR (BIT)	11	20.5	70.5	161.5	4	5.5	12.5	27.5	0.3	0.7	3.1	6.1	24

Note:
1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available at print time for this manual (represented by a dash: –), refer to the IPI for that CPU.
2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/the number of bits or words.
3. Enabled time is for single length units of type %R.
4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
5. DOIO is the time to output values to discrete output module.

Table A-1. Instruction Timing - Continued

Function Group	Function	Enabled 924/925	Enabled 914/915	Enabled 781/782 788/789	Enabled 731/732 771/772	Disabled 924/925	Disabled 914/915	Disabled 781/782 788/789	Disabled 731/732 771/772	Increment 924/925	Increment 914/915	Increment 781/782 788/789	Increment 731/732 771/772	Size
Data Move	MOVE (WORD)	4.5	5.5	14.5	29.5	3.5	3.5	11.5	18.5	0.05	0.1	0.4	0.8	15
	MOVE (DWORD)	7.5	9.5	32.5	56.5	5.5	6.5	18	31	0.2	0.2	1	1.7	15
	BLKMOV (WORD)	5.5	8	21	36	3	3.5	7.5	16.5	-	-	-	-	30
	BLKMOV (DWORD)	7	10	28.5	55	6.5	9	28.5	50.5	-	-	-	-	44
	SWAP (WORD)	5.5	7.5	25	45.5	3	3.5	12	17.5	0.3	0.6	1.7	3.5	15
	SWAP (DWORD)	7	10.5	36.5	65	5	6.5	20.5	31	0.5	0.9	2.7	5.6	15
	BLKCLR	4.5	6	19	30.5	3	3.5	10.5	16.5	0.1	0.2	0.6	1.4	12
	BITSEQ	10	15	46.5	95	7	11.5	38.5	80.5	-	-	-	-	24
	SHFR (WORD)	11	15.5	54	113.5	4	5	11.5	28.5	0.05	0.05	0.3	0.6	24
	SHFR (DWORD)	11.5	16.5	54	117	5	6	14.5	30.5	0.1	0.1	0.8	1.3	24
	SORT	258	423	1403.5	2894	5	6	18.5	28	2.02	3.61	14.43	36.92	15
Data Table	TBLRD (INT)	5.5	8.5	27.5	53.5	3.5	4	11	17.5	-	-	-	-	21
	TBLRD (DINT)	7.5	11.5	28.5	58.5	4	4	12.5	18	-		-	-	21
	TBLWR (INT)	6	9	30	60.5	3.5	3.5	11.5	17.5	-		-	-	21
	TBLWR (DINT)	8.5	14	52	104.5	6	8.5	19.5	37.5	-		-	-	21
	FIFORD (INT)	5.5	9	29	58.5	2.5	3	9.5	17	-		-	-	21
	FIFORD (DINT)	7	11	31.5	61	3.5	4	12	18	-		-	-	21
	FIFOWRT (INT)	5.5	8	25.5	55	3	4	11.5	17.5	-		-	-	21
	FIFOWRT (DINT)	7	13	45	85	5	7.5	21.5	37.5	-		-	-	21
	LIFORD (INT)	6	9	27.5	52	2.5	3.5	11.5	17	-		-	-	21
	LIFORD (DINT)	7.5	12	29.5	57	3.5	4	11.5	18	-		-	-	21
	LIFOWRT (INT)	5.5	8	30	55	2.5	3.5	11.5	17	-		-	-	21
	LIFOWRT (DINT)	8	14	44.5	85	6	9	23.5	37.5	-		-	-	21
	ARRAY_MOVE (BIT)	18.5	27.5	91	197.5	5.5	9	23	41.5	0.01	0.01	0.10	0.30	–
	ARRAY_MOVE (BYTE)	12	20.5	74	163	6	9.5	23	42.5	0.10	0.21	0.50	1.20	–
	ARRAY_MOVE (WORD)	12	20.5	76	164	5.5	9.5	23	41.5	0.03	0.04	0.10	0.30	–
	ARRAY_MOVE (DWORD)	12	20.5	76	165	6.5	9.5	25.5	43	0.09	0.09	0.30	0.60	–
	SRCH (BYTE)	9.5	15	56	119	5.5	8.5	25	40.5	0.03	0.07	0.30	0.60	–
	SRCH (WORD)	9.5	15	57	122.5	5.5	8.5	25.5	41	0.04	0.08	0.30	0.60	–
	SRCH (DWORD)	10	16	60	132	6	8.5	24	41	0.12	0.25	0.80	1.50	–
	ARRAY_RANGE (WORD)	12	16.5	60.5	124	5.5	8	22.5	38	0.24	0.46	2	4.80	–
	ARRAY_RANGE (DWORD)	11.5	16.5	57.5	127.5	7.5	8	20	38	0.25	0.50	2.10	4.80	–
Conversion	to INT (UINT)	4.5	6.5	21.5	32	3	3.5	10.5	16.5	-	-	-	-	12
	to INT (DINT)	5.5	7	23	40	4.5	5.5	16	26	-	-	-	-	12
	to INT (BCD-4)	5	7	20	38.5	3.5	3.5	9	17	-	-	-	-	12
	to DINT (INT)	6.5	7.5	21	33	4.5	5	11	17.5	-	-	-	-	12
	to DINT (UINT)	5.5	7	23.5	32.5	4	4.5	11	17.5	-	-	-	-	12
	to DINT (BCD-8)	11.5	13	44.5	76.5	5	6	19.5	26.5	-	-	-	-	12
	to UINT (INT)	4.5	6	16.5	32	4	5	7.5	17	-	-	-	-	12
	to UINT (DINT)	5.5	6.5	19.5	39.5	5	5.5	15	26	-	-	-	-	12
	to UINT (BCD-4)	5	7.5	20	38.5	3.5	3.5	7.5	17	-	-	-	-	12
	to BCD-4 (INT)	7	9.5	29	44	3.5	3.5	16	16.5	-	-	-	-	12
	to BCD-4 (UINT)	5.5	7.5	25	43	3	3.5	12	16.5	-	-	-	-	12
	to BCD-8 (DINT)	8.5	13	38.5	77	5	6	18.5	26.5	-	-	-	-	12

Note:
1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available when this manual was being printed (represented by a dash: –), refer to the IPI for each CPU.
2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/the number of bits or words.
3. Enabled time is for single length units of type %R.
4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
5. DOIO is the time to output values to discrete output module.

Table A-1. Instruction Timing - Continued

Function Group	Function	Enabled 924/925	Enabled 914/915	Enabled 781/782 788/789	Enabled 731/732 771/772	Disabled 924/925	Disabled 914/915	Disabled 781/782 788/789	Disabled 731/732 771/772	Increment 924/925	Increment 914/915	Increment 781/782 788/789	Increment 731/732 771/772	Size
Control	JUMP	2.5	2.5	6.5	13	2	2	2.5	5	-	-	-	-	–
	FOR/NEXT	13.5	19.5	57	124	5.5	8.5	22	52.5	-	-	-	-	–
	MCR/ENDMCR Combined	7	11	28	51	6	9.5	28	46.5	-	-	-	-	9
	DOIO	25.5	40	122	243.5	3.5	4.5	11	20.5	-	-	-	-	15
	DOIO with ALT	32.5	47	140.5	311.5	4	5	11.5	20.5	-	-	-	-	15
	SUSIO	3.5	4.5	12	22	3	4	7	15.5	-	-	-	-	6
	COMMREQ	113	137	350	790	9	11	22.5	38.5	-	-	-	-	18
	CALL/RETURN (LD)	23	31	90.5	229.5	2.5	2.5	5	6	-	-	-	-	15
	CALL/RETURN (SFC)	104.5	143	390	731.5	2.5	2.5	5	6	-	-	-	-	15
	CALL/RETURN (PSB)	16.5	23.5	73.5	145	3	3	3.5	7	-	-	-	-	15
	CALL/RETURN (External Block)	42.5	74	262.5	309.5	2.5	2.5	3	6	-	-	-	-	15
	PIDISA	6.5	11	40	75.5	6	9.5	33.5	65	-	-	-	-	27
	PIDIND	6	11	41.5	76.5	6.5	10	31	66.5	-	-	-	-	27
	VMERD (BYTE)	16	27.5	89.5	173.5	4.5	7.5	20.5	35	0.8	0.8	0.9	1.2	18
	VMERD (WORD)	16	27.5	89.5	171.5	4.5	7.5	19	34	0.8	0.8	0.9	1.2	18
	VMEWRT (BYTE)	19	30	102.5	192.5	5.5	8	20	36.5	0.8	0.8	0.8	0.9	18
	VMEWRT (WORD)	19	30	97.5	188	4.5	7.5	19.5	34	0.8	0.8	0.8	0.9	18
	VMERMW	21	33.5	99.5	191.5	4.5	7.5	20.5	35	-	-	-	-	18
	VMETST	21	28.5	86	155.5	6.5	9.5	25.5	39	-	-	-	-	15
	VME_CFG_RD	31.5	50.5	170.5	345.5	5.5	9.5	26.5	45	-	-	-	-	–
	VME_CFG_WRT	26.5	43	159.5	333.5	5.5	8.5	28	45	-	-	-	-	–
	SVCREQ:													12
	#1	12	23	79	97	4.5	6	14	26	-	-	-	-	12
	#2	12	21	77	86	4.5	6	14	26	-	-	-	-	12
	#3	12	22	76	85	4.5	6	14	26	-	-	-	-	12
	#4	11	21	74	85	4.5	6	14	26	-	-	-	-	12
	#6	13	22	82	93	4.5	6	14	26	-	-	-	-	12
	#7	24	41	152	288	4.5	6	14	26	-	-	-	-	12
	#8	20	37	143	279	4.5	6	14	26	-	-	-	-	12
	#9	23	36	130	202	4.5	6	14	26	-	-	-	-	12
	#10	15	25	90	127	4.5	6	14	26	-	-	-	-	12
	#11	12	22	82	117	4.5	6	14	26	-	-	-	-	12
	#12	11	20	77	82	4.5	6	14	26	-	-	-	-	12
	#13	-	-	-	-	-	-	-	-	-	-	-	-	12
	#14	150	203	643	963	4.5	6	14	26	-	-	-	-	12
	#15	15	25	91	156	4.5	6	14	26	-	-	-	-	12
	#16	19	31	115	148	4.5	6	14	26	-	-	-	-	12
	#17	23	40	94	346	4.5	6	14	26	-	-	-	-	12
	#18	2939	2998	3199	1203	4.5	6	14	26	-	-	-	-	12
	#19	34	52	77	92	4.5	6	14	26	-	-	-	-	12
	#20	26	42	128	248	4.5	6	14	26	-	-	-	-	12
	#21	138	178	586	1558	4.5	6	14	26	-	-	-	-	12
	#22	11	21	77	85	4.5	6	14	26	-	-	-	-	12

Note:
1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available when this manual was being printed (represented by a dash: –), refer to the IPI for each CPU.
2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/the number of bits or words.
3. Enabled time is for single length units of type %R.
4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
5. DOIO is the time to output values to discrete output module.

Table A-1. Instruction Timing - Continued

Function Group	Function	Enabled				Disabled				Increment				Size
		924/ 925	914/ 915	781/782 788/789	731/732 771/772	924/ 925	914/ 915	781/782 788/789	731/732 771/772	924/ 925	914/ 915	781/782 788/789	731/732 771/772	
Floating Point	**Math**:													
	ADD_REAL	8.5	13.5	52.5	108	5.5	6.5	19	31	-	-	-	-	–
	SUB_REAL	8.5	13	51.5	108	5.5	6.5	22.5	31	-	-	-	-	–
	MUL_REAL	8.5	13	51	108	5.5	6.5	21	31	-	-	-	-	–
	DIV_REAL	9.5	14	54	109	5.5	7	22	31	-	-	-	-	–
	ABS_REAL	7	11	42.5	85	5	6.5	17	26.5	-	-	-	-	–
	SQRT_REAL	8	13	43.5	90.5	6	7	14	26.5	-	-	-	-	–
	Trigonometric:													
	SIN	10.5	20.5	63.5	126.5	3.5	6	16	25.5	-	-	-	-	–
	COS	10	20	61.5	121.5	2.5	5.5	14	25.5	-	-	-	-	–
	TAN	9	18	58.5	114.5	3	5.5	14	25	-	-	-	-	–
	ASIN	6.5	13.5	59	132	2.5	5.5	12.5	25.5	-	-	-	-	–
	ACOS	8.5	17.5	69.5	165	2.5	5.5	14.5	25	-	-	-	-	–
	ATAN	10.5	19.5	54	110.5	3.5	5.5	16	25	-	-	-	-	–
	Logarithmic:													
	LOG	6	13	48	100.5	2.5	5.5	13.5	25.5	-	-	-	-	–
	LN	7	13.5	47.5	99.5	2.5	5.5	15.5	25	-	-	-	-	–
	EXPT	10.5	21.5	92	228	3	6	17.5	30	-	-	-	-	–
	EXP	11	22.5	90.5	194	2.5	5.5	14	25.5	-	-	-	-	–
	Comparison:													
	EQ_REAL	7.5	11.5	40	87	5	6	17.5	31	-	-	-	-	–
	NE_REAL	7	10.5	41	85	5	6	20	31	-	-	-	-	–
	GT_REAL	7.5	11.5	46	92	5	6	15	31	-	-	-	-	–
	GE_REAL	7.5	11	46	92	5	6	13.5	30	-	-	-	-	–
	LT_REAL	8	11	43.5	90	5.5	8	19.5	31	-	-	-	-	–
	LE_REAL	7.5	11	45	90	5	6	21.5	31	-	-	-	-	–
	CMP_REAL	17	20	51	114	13	16.5	35.5	64.5	-	-	-	-	–
	Data Move:													
	MOVE_REAL	6	9	29	55.5	3.5	6	14	30	-	-	-	-	–
	Conversion:													
	REAL_TO_INT	7	12.5	43	96.5	4	6	17	26	-	-	-	-	–
	REAL_TO_UINT	8.5	13	46.5	112.5	4	6	16	25.5	-	-	-	-	–
	REAL_TO_DINT	8	13.5	41.5	94	5	7	14.5	25.5	-	-	-	-	–
	INT_TO_REAL	6	10	36.5	68	4	6	14.5	25	-	-	-	-	–
	UINT_TO_REAL	7.5	10.5	30	70.5	4	6	11.5	26	-	-	-	-	–
	DINT_TO_REAL	5.5	9.5	28	65.5	5	5.5	14	25.5	-	-	-	-	–
	REAL_TRUN_INT	8.5	13.5	50	126.5	5	7	15.5	26	-	-	-	-	–
	REAL_TRUN_DINT	8	13.5	51.5	121.5	4	6.5	14.5	26	-	-	-	-	–
	DEG_TO_RAD	9	14.5	43.5	101	5.5	7	12	26	-	-	-	-	–
	RAD_TO_DEG	7.5	14	46	105	4	6	12	26	-	-	-	-	–
	BCD4_TO_REAL	7.5	13	36	92	4	5.5	15	25.5	-	-	-	-	–
	BCD8_TO_REAL	9.5	15	39.5	100	4	6	13.5	26	-	-	-	-	–

Note:
1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available when this manual was being printed (represented by a dash: –), refer to the IPI for each CPU.
2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/the number of bits or words.
3. Enabled time is for single length units of type %R.
4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
5. DOIO is the time to output values to discrete output module.

Table A-1. Instruction Timing - Continued

Function Group	Function	Enabled			Disabled			Increment			Size
		935	928	CPX782 CPX772	935	928	CPX782 CPX772	935	928	CPX782 CPX772	
Timers	ONDTR	5.5	6	23.5	5	5	16.5	-	-	-	18
	OFDT	5.5	5.5	21	5.5	5.5	20	-	-	-	15
	TMR	6	6.5	23.5	5.5	6	20.5	-	-	-	15
Counters	UPCTR	5.5	5.5	19	5	5	18	-	-	-	18
	DNCTR	5.5	5.5	19	5	5	18	-	-	-	18
Math	ADD (INT)	4	4	10.5	3.5	3.5	7.5	-	-	-	15
	ADD (DINT)	5.5	6	20	4.5	5	11.5	-	-	-	15
	SUB (INT)	4	4	10.5	3.5	3.5	7.5	-	-	-	15
	SUB (DINT)	5.5	6	20	4.5	4.5	11	-	-	-	15
	MUL (INT)	4	4	11	3.5	3.5	8	-	-	-	15
	MUL (DINT)	6	6	21	4.5	4.5	11.5	-	-	-	15
	DIV (INT)	4.5	4.5	11.5	3.5	3.5	8	-	-	-	15
	DIV (DINT)	6.5	6.5	21	5.5	5.5	11.5	-	-	-	15
	MOD (INT)	4.5	4.5	12	3.5	3.5	8	-	-	-	15
	MOD (DINT)	7	7	25	5	5	12	-	-	-	15
	ABS (INT)	4.5	5	17	3.5	4	9.5	-	-	-	12
	ABS (DINT)	5.5	5.5	17	4.5	4.5	10	-	-	-	12
	SQRT (INT)	6.5	6.5	20	3	3	6.5	-	-	-	12
	SQRT (DINT)	9	9	37.5	4.5	4.5	10	-	-	-	12
Relational	EQ (INT)	5.5	5.5	14	3	3.5	7	-	-	-	15
	EQ (DINT)	6	6	19.5	4.5	4.5	11	-	-	-	15
	NE (INT)	5.5	5.5	13	3	3.5	7	-	-	-	15
	NE (DINT)	6	6	19	4.5	4.5	11	-	-	-	15
	GT (INT)	6.5	6.5	16	3	3.5	7	-	-	-	15
	GT (DINT)	7	7	20.5	4.5	4.5	12	-	-	-	15
	GE (INT)	5.5	5.5	15.5	3	3.5	7	-	-	-	15
	GE (DINT)	6	6	19.5	4.5	4.5	11	-	-	-	15
	LT (INT)	6.5	6.5	17	3	3.5	7.5	-	-	-	15
	LT (DINT)	7	7	20.5	4.5	4.5	11.5	-	-	-	15
	LE (INT)	5.5	5.5	15.5	3	3.5	7	-	-	-	15
	LE (DINT)	6	6	20	4.5	4.5	11	-	-	-	15
	CMP (INT)	9.5	9.5	22.5	7.5	8	20.5	-	-	-	21
	CMP (DINT)	10	10.5	27.5	8	8	21	-	-	-	21
	RANGE (INT)	7	7.5	28	4.5	4.5	13	-	-	-	
	RANGE (DINT)	7.5	7.5	30.5	4.5	5	13.5	-	-	-	
Bit Operation	AND (WORD)	6.5	7	32	4.5	4.5	13	0.5	0.5	5.1	18
	AND(DWORD)	7.5	7.5	34	5	5	13.5	0.6	0.7	6	18
	OR (WORD)	6.5	7	32.5	4.5	4.5	13	0.5	0.5	5.4	18
	OR (DWORD)	7.5	8	35.5	5	5	14	0.7	0.8	7.3	18
	XOR (WORD)	6.5	7	32.5	4.5	4.5	13.5	0.5	0.5	5.4	18
	XOR (DWORD)	7.5	7.5	34.5	5	5	13.5	0.7	0.8	7	18
	NOT (WORD)	5.5	5.5	23.5	4	4	12	0.2	0.2	1.9	15
	NOT (DWORD)	6	6	23.5	5	5	12	0.2	0.2	1.6	15
	MCMP (WORD)	9.5	10	58.5	6.5	6.5	18	0.4	0.4	3.3	30
	MCMP (DWORD)	10.5	11	62	7	7	18	0.8	0.9	6.7	30
	SHL (WORD)	9.5	9.5	37.5	4.5	4.5	13.5	0.4	0.5	2.3	24
	SHL (DWORD)	10.5	10.5	40	5.5	5.5	14.5	0.9	1.1	4.8	24
	SHR (WORD)	8.5	8.5	39	4.5	4.5	15	0.4	0.5	2.3	24
	SHR (DWORD)	9.5	9.5	41	5.5	5.5	15.5	0.9	1.1	4.7	24
	ROL (WORD)	7.5	7.5	32	4.5	4.5	12.5	0.4	0.5	2.3	18
	ROL (DWORD)	8	8	34.5	5	5	13	1	1.1	4.8	18
	ROR (WORD)	6.5	6.5	31.5	4.5	4.5	12.5	0.5	0.5	2.5	18
	ROR (DWORD)	8	8	34	5	4.5	14	1	1.1	5.1	18
	BTST (WORD)	6.5	6.5	26	4.5	4.5	12.5	-	-	-	18
	BTST (DWORD)	6.5	6.5	25	4.5	4.5	11.5	-	-	-	18
	BSET (WORD)	5.5	5.5	25	4	4	11	-	-	-	15
	BSET (DWORD)	5.5	5.5	23.5	4.5	4.5	9.5	-	-	-	15
	BCLR (WORD)	5.5	5.5	24.5	4	4	11	-	-	-	15
	BCLR (DWORD)	5.5	5.5	24.5	4	4	11	-	-	-	15
	MOVE (BIT)	11	11	36	4	4	11.5	0.7	0.7	1.8	15
	BPOS (WORD)	6.5	6.5	30	4	4	11	0.1	0.1	1.3	18
	BPOS (DWORD)	8.5	8.5	40.5	4	4	11	0.2	0.2	2.3	18
	SHFR (BIT)	8	8	43	3.5	3.5	8.5	0.2	0.2	1.8	24

Note: 1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available at print time for this manual (represented by a dash; –), refer to the IPI for that CPU.
2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/the number of ts or rds.
3. Enabled time is for single length units of type %R.
4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
5. DOIO is the time to output values to discrete output module.

Table A-1. Instruction Timing - Continued

Function Group	Function	Enabled			Disabled			Increment			Size
		935	928	CPX772 CPX782	935	928	CPX772 CPX782	935	928	CPX772 CPX782	
Data Move	MOVE (WORD)	4	4	9.5	3	3	7	0.1	0.1	0.3	15
	MOVE (DWORD)	5.5	5.5	18.5	4.5	4.5	11	0.2	0.2	0.8	15
	BLKMOV (WORD)	4.5	4.5	13	2.5	2.5	5	-	-	-	30
	BLKMOV (DWORD)	6	6	18	5.5	5.5	19	-	-	-	44
	SWAP (WORD)	4.5	4.5	14	2.5	2.5	6.5	0.2	0.2	1.3	15
	SWAP (DWORD)	6	6	20.5	4.5	4.5	11	0.3	0.3	1.5	15
	BLKCLR	4	4	10.5	2.5	2.5	5.5	0.1	0.2	0.3	12
	BITSEQ	9	9	28.5	6	6	24	-	-	-	24
	SHFR (WORD)	7	7	32.5	3.5	3.5	9	0.1	0.1	0.3	24
	SHFR (DWORD)	8	8	34	4.5	4.5	10	0.2	0.2	0.7	24
	SORT	215.5	260	788	4.5	4	13	1.38	1.53	8.8	15
Data Table	TBLRD (INT)	4	4	15.5	2	2	4.5	-	-	-	21
	TBLRD (DINT)	5	5	17	3	3	5.5	-	-	-	21
	TBLWR (INT)	4.5	4.5	17.5	2	2	5	-			21
	TBLWR (DINT)	6.5	6.5	31.5	4	4	12.5	-	-	-	21
	FIFORD (INT)	4	4	16.5	2	2	5.5	-	-	-	21
	FIFORD (DINT)	5	5	18	3	3	5.5	-	-	-	21
	FIFOWRT (INT)	4	4	15	2	2	5	-	-	-	21
	FIFOWRT (DINT	5.5	5.5	25.5	4	4	13	-	-	-	21
	LIFORD (INT)	4	4	14	2	2	5	-	-	-	21
	LIFORD (DINT)	5	5	17	3	3	6	-	-	-	21
	LIFOWRT (INT)	4	4	15	2	2	5	-	-	-	21
	LIFOWRT (DINT)	5.5	5.5	26	4.5	4.5	13.5	-	-	-	21
Array	ARRAY_MOVE (BIT)	9.5	10	52.5	5	5	14.5	0.01	0.01	0.07	–
	ARRAY_MOVE (BYTE)	8.5	9	44	5	5	15	0.07	0.07	0.40	–
	ARRAY_MOVE (WORD)	8.5	9	45.5	5	5	15.5	0.04	0.06	0.10	–
	ARRAY_MOVE (DWORD)	9.5	9.5	46	5.5	5.5	16	0.09	0.12	0.30	–
	SRCH (BYTE)	8	8	36	5	5	14.5	0.02	0.02	0.10	–
	SRCH (WORD)	8	8	35.5	5	5	14.5	0.02	0.02	0.10	–
	SRCH (DWORD)	8	8.5	38.5	5	5	14.5	0.08	0.08	0.40	–
	ARRAY_RANGE (WORD)	7.5	7.5	38	5	5	14	0.15	0.16	1.30	–
	ARRAY_RANGE (DWORD)	7.5	7.5	37.5	5	5	14	0.16	0.17	1.30	–
Conversion	to INT (UINT)	4	4	11.5	3	3	5.5	-	-	-	12
	to INT (DINT)	5	5	13.5	4	4	9.5	-	-	-	12
	to INT (BCD-4)	4.5	4.5	12	3	3	6	-	-	-	12
	to DINT (INT)	5	5	12	3.5	3.5	5.5	-	-	-	12
	to DINT (UINT)	5	5	11.5	3.5	3.5	6.5	-	-	-	12
	to DINT (BCD-8)	6.5	6.5	19.5	4.5	4	9.5	-	-	-	12
	to UINT (INT)	4	4	10.5	3	3	6.5	-	-	-	12
	to UINT (DINT)	5	5	13.5	4	4	9	-	-	-	12
	to UINT (BCD-4)	4.5	4.5	12.5	3	3	6.5	-	-	-	12
	to BCD-4 (INT)	4.5	4.5	12	3	3	6	-	-	-	12
	to BCD-4 (UINT)	4.5	4.5	13	3	3	6	-	-	-	12
	to BCD-8 (DINT)	7	7	18	4.5	4.5	9.5	-	-	-	12

Note: 1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available when this manual was being printed (represented by a dash: –), refer to the IPI for each CPU.
2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/the number of bits or words.
3. Enabled time is for single length units of type %R.
4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
5. DOIO is the time to output values to discrete output module

Table A-1. Instruction Timing - Continued

Function Group	Function	Enabled			Disabled			Increment			Size
		935	**928**	**CPX772 CPX782**	**935**	**928**	**CPX772 CPX782**	**935**	**928**	**CPX772 CPX782**	**Size**
Control	JUMP	2.5	2.5	3.5	2	2	3	-	-	-	–
	FOR/NEXT	10.5	11	38	5	5	16.5	-	-	-	–
	MCR/ENDMCR Combined	6	6.5	18.5	5	5.5	17.5	-	-	-	9
	DOIO	13.5	15	81.5	3	3	7.5	-	-	-	15
	DOIO with ALT	14	16	95.5	3	3	7.5	-	-	-	15
	SUSIO	3	3	7.5	2.5	2.5	6	-	-	-	6
	COMMREQ	84.5	90	240.5	8.5	8.5	16	-	-	-	18
	CALL/RETURN (LD)	23	24.5	57.5	2	2	2.5	-	-	-	15
	CALL/RETURN (SFC)	82	88	251	2	2	2.5	-	-	-	15
	CALL/RETURN (PSB)	15.5	17.5	47.5	3	3	3.5	-	-	-	15
	CALL/RETURN (External Block)	27.5	31.5	152.5	2	2	2.5	-	-	-	15
	PIDISA	6	6	22.5	6	6	20.5	-	-	-	27
	PIDIND	6	6	23	6	6	20.5	-	-	-	27
	VMERD (BYTE)	13.5	13.5	53	4	4	13	0.8	0.8	0.9	18
	VMERD (WORD)	13.5	13.5	50.5	4	4	12	0.8	0.8	0.9	18
	VMEWRT (BYTE)	14.5	14.5	55	4	4	13	0.8	0.8	0.8	18
	VMEWRT (WORD)	14.5	14.5	54.5	4	4	12.5	0.8	0.8	0.8	18
	VMERMW	15	15.5	56.5	4	4	13	-	-	-	18
	VMETST	17	17	49	6	6	14.5	-	-	-	15
	VME_CFG_RD	26	26.5	104	4.5	4.5	16.5	-	-	-	–
	VME_CFG_WRT	17.5	18	93	4.5	4.5	16	-	-	-	–
	SVCREQ:										12
	#1	9	9	49	3.5	3.5	10.5	-	-	-	12
	#2	9	9	49	3.5	3.5	10.5	-	-	-	12
	#3	9	9	48	3.5	3.5	10.5	-	-	-	12
	#4	8	9	48	3.5	3.5	10.5	-	-	-	12
	#6	11	11	50	3.5	3.5	10.5	-	-	-	12
	#7	19	21	96	3.5	3.5	10.5	-	-	-	12
	#8	22	25	120	3.5	3.5	10.5	-	-	-	12
	#9	19	19	71	3.5	3.5	10.5	-	-	-	12
	#10	11	11	54	3.5	3.5	10.5	-	-	-	12
	#11	10	10	50	3.5	3.5	10.5	-	-	-	12
	#12	9	9	48	3.5	3.5	10.5	-	-	-	12
	#13	-	-	-	-	-	-	-	-	-	12
	#14	128	159	370	3.5	3.5	10.5	-	-	-	12
	#15	10	10	55	3.5	3.5	10.5	-	-	-	12
	#16	16	17	64	3.5	3.5	10.5	-	-	-	12
	#17	14	15	91	3.5	3.5	10.5	-	-	-	12
	#18	2985	2986	546 (772) 3025 (782)	3.5	3.5	10.5	-	-	-	12
	#19	24	24	112	3.5	3.5	10.5	-	-	-	12
	#20	20	23	99	3.5	3.5	10.5	-	-	-	12
	#21	77	91	381	3.5	3.5	10.5	-	-	-	12
	#22	9	9	47	3.5	3.5	10.5	-	-	-	12

<u>Note</u> 1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available when this manual was being printed (represented by a dash: –), refer to the IPI for each CPU.
   2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/ the number of bits or words.
   3. Enabled time is for single length units of type %R.
   4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
   5. DOIO is the time to output values to discrete output module.

Table A-1. Instruction Timing - Continued

Function Group	Function	Enabled			Disabled			Increment			Size
		935	**928**	**CPX772 CPX782**	**935**	**928**	**CPX772 CPX782**	**935**	**928**	**CPX772 CPX782**	
Floating Point	**Math**:										
	ADD_REAL	7	7	21.5	4.5	4.5	11	-	-	-	–
	SUB_REAL	7	7	21	4.5	4.5	11	-	-	-	–
	MUL_REAL	7	7	21.5	4.5	4.5	11	-	-	-	–
	DIV_REAL	7	7	21.5	4.5	4.5	12	-	-	-	–
	ABS_REAL	6	6	17.5	4.5	4.5	10	-	-	-	–
	SQRT_REAL	6.5	6.5	18	4.5	4.5	10.5	-	-	-	–
	Trigonometric:										
	SIN	8	8	19.5	3	3	9	-	-	-	–
	COS	8	8	20	3	3	9	-	-	-	–
	TAN	7.5	7.5	19.5	3	3	9.5	-	-	-	–
	ASIN	6	6	17.5	3	3	9.5	-	-	-	–
	ACOS	7	7	19	3	3	9.5	-	-	-	–
	ATAN	8	8	20	3	3	9.5	-	-	-	–
	Logarithmic:										
	LOG	5.5	5.5	19	3	3	9	-	-	-	–
	LN	5.5	5.5	18.5	3	3	9	-	-	-	–
	EXPT	8.5	8.5	27	3	3	10.5	-	-	-	–
	EXP	9	9	20	3	3	9	-	-	-	–
	Comparison:										
	EQ_REAL	7	7	21	5	5	12	-	-	-	–
	NE_REAL	7	7	19.5	5	5	11.5	-	-	-	–
	GT_REAL	7	7	20.5	5	5	11.5	-	-	-	–
	GE_REAL	7	7	20.5	5	5	11.5	-	-	-	–
	LT_REAL	7	7	19.5	5	5	11.5	-	-	-	–
	LE_REAL	7	7	19.5	5	5	11	-	-	-	–
	CMP_REAL	16.5	16.5	32	13.5	13.5	25.5	-	-	-	–
	Data Move:										
	MOVE_REAL	4.5	4.5	17.5	3.5	3.5	10	-	-	-	–
	Conversion:										
	REAL_TO_INT	6	6	18	3.5	3.5	9.5	-	-	-	–
	REAL_TO_UINT	6	6	20	3.5	3.5	9.5	-	-	-	–
	REAL_TO_DINT	5.5	5.5	18	3.5	3.5	9.5	-	-	-	-
	INT_TO_REAL	5	5	17	3.5	3.5	9.5	-	-	-	–
	UINT_TO_REAL	5	5	17.5	3.5	3.5	9.5	-	-	-	-
	DINT_TO_REAL	4.5	4.5	16	3.5	3.5	9.5	-	-	-	–
	REAL_TRUN_INT	6	6	20.5	3.5	3.5	10	-	-	-	–
	REAL_TRUN_DINT	6	6	20	3.5	3.5	9	-	-	-	–
	DEG_TO_RAD	5.5	5.5	17.5	3.5	3.5	9.5	-	-	-	–
	RAD_TO_DEG	6	6	18.5	3.5	3.5	9.5	-	-	-	–
	BCD4_TO_REAL	6	6	19.5	3.5	3.5	9.5	-	-	-	–
	BCD8_TO_REAL	6.5	6.5	21.5	3.5	3.5	9	-	-	-	–

Note: 1. Time (in microseconds) is based on Release 5.0 of Logicmaster 90-70 software. For information not available when this manual was being printed (represented by a dash: –), refer to the IPI for each CPU.
2. For table functions, increment is in units of length specified. For bit operation functions, microseconds/bit. For data move functions, microseconds/the number of bits or words.
3. Enabled time is for single length units of type %R.
4. COMMREQ time has been measured between CPU and EX7 with NOWAIT option.
5. DOIO is the time to output values to discrete output module.

Overhead Sweep Impact Time

This part of the appendix contains overhead timing information for the Series 90-70 PLC CPU. This information can be used in conjunction with the estimated logic execution time to predict sweep times for each of the Series 90-70 CPUs. The information in this section is made up of a base sweep time plus sweep impact times for each of the CPU models: 731, 732, 771, 772, 781, 782, 914, 915, 924,925, 928 and 935. The predicted sweep time is computed by adding the sweep impact time(s), the base sweep, and the estimated logic execution time.

Two examples of predicting sweep times are provided at the end of this appendix.

Sweep impact times are composed of four basic sections:

1. Programmer communications sweep impact

2. I/O Scan and fault sweep impact

3. Ethernet Global Data sweep impact

4. Intelligent Option Module (PCMs and LAN modules) sweep impact

5. I/O interrupt performance and sweep impact

Each of these sections describes the functions and provides tables with the corresponding times for each CPU model.

The information in these tables may be used to predict sweep time based on a given configuration.

What the Tables Contain

The following tables contain sweep impact times for overhead functions for the Series 90-70 PLC. 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 rest of the timing values are given as sweep impact times, that is, the time added to the sweep by the function in question. Sweep impact times are nominal.

Note

There are two categories of sweep impact numbers listed in the tables—those that impact the sweep every sweep and those that impact the sweep only when invoked. The functions that impact the sweep every sweep are listed in bold type in each table.

In some of the 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 PLC 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 Programmer Communications Window and the System 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 timeslicing (limiting) of the window occurs in a given PLC 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 timeslice over successive sweeps.

Base Sweep Times

The base sweep time for each CPU model is shown below. This time is for an empty _MAIN program block with no programmer attached, no configuration downloaded, and no other module present in the system other than the CPU. The following diagram shows the full sweep phases and the base sweep phases contrasted so that the optional parts of the sweep are illustrated.

Table A-2. 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 **
↓	↓
↓	Programmer Communications Window
↓	↓
↓	System Communications Window
<END OF SWEEP>	<END OF SWEEP>

*   If I/O is suspended, then the input and output scans are skipped.
**   Polling for missing I/O modules only occurs if a "Loss of ..." fault has been logged for a Series 90-70 I/O module.
***   If no Ethernet Global Data (EGD) exchanges are configured, then the consumption and production scans are skipped.

For the base sweep, the lack of configuration means that 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, PSM, etc.) would have 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 Programmer Communications Window is never opened. The lack of intelligent option modules means that the System Communications Window is never opened.

The following table gives the base sweep times in milliseconds for each CPU model.

Table A-3. Base Sweep Times

CPU Model	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
Base Sweep Time	.3731	.4001	.1525	.2074	1.0687	.7025	1.0687	2.00

Programmer Sweep Impact Times

The following table shows the programmer sweep impact times in milliseconds. The times are broken up into parallel and serial since these interfaces are significantly different. (Each item in the table is described in more detail at the end of the table.)

Table A-4. Programmer Sweep Impact Times*

Sweep Impact Item	CPX 935	CPX 928	914/ 925	924/ 925	CPX 782	781/782 788/789	CPX 772	731/731 771/772
Parallel Programmer								
Programmer window	-	-	-	-	-	-	-	0.75
Reference table monitor	-	-	-	-	-	-	-	4.90
Editor monitor	-	-	-	-	-	-	-	4.90
Word-for-word change	-	-	-	-	-	-	-	28.10
ALT-S store	-	-	-	-	-	-	-	14.80
High priority request	-	-	-	-	-	-	-	9.90
Serial programmer								
Programmer Window	-	-	-	-	-	-	-	0.50
Reference table monitor	-	-	-	-	-	-	-	5.60
Editor monitor	-	-	-	-	-	-	-	6.40

* Not all of the timing information needed for the above table was available at print time for this manual (the dashes).

Note

Functions in bold type in Table A-4 above impact the sweep continuously. All other functions impact the sweep only when invoked.

Each of the items included in the table is described below.

Sweep Impact Item	Description
Programmer Window	**<u>Parallel:</u>** The time to open the Programmer Window but not process any requests. The programmer is attached with a parallel connection; no reference values are being monitored.
	<u>Serial:</u> Same as above except attached serially. Unlike the parallel programmer, the sweep impact cannot be limited to a single sweep. Serial communications, in addition to the service processing time that is performed inside the Programmer Communications Window, has processing time associated with the receiving of data over the RS422/485 serial link. This serial communications processing time is asynchronous to the sweep in order to meet the timing requirements of the serial protocol.
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 PLC and the speed of the host of the programming software.
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.
Word-for-Word Change	The sweep impact to change a constant input on a MOVE_UINT function from 1 to 2. This is the smallest change that can be made. A change to a coil requires updates to the coil use and retentive maps contained in the PLC. If the %Q or %M reference address is changed on a function block, then the size of the change to the coil use and retentive maps can be quite large. A large word-for-word change will have very little sweep impact if done with the parallel programmer (worst-case of 35 ms on a 731 CPU). A large word-for-word change will have a big sweep impact if done with the serial programmer. The time is linear for the number of bytes in the request with the worst case being slightly larger than the worst case for parallel.
ALT-S Store	The sweep impact to store a 933 byte program block. The sweep impact is linear as the program block gets larger.
High Priority Request (Parallel only)	There are several programmer high priority requests: change PLC mode (includes ALT-R), read constant sweep state/time, read window times, change constant sweep state/time, and change window times. Unlike the other programmer service requests, which are serviced inside the programmer communications window, high priority requests interrupt the sweep and are serviced asynchronously at the time they are issued by the programmer. The worst case high priority request is monitoring/changing the window times. This time is shown below.

I/O Scan and I/O Fault Sweep Impact

The I/O scan sweep impact has two parts, Series 90-70 I/O and Genius I/O. The equation for computing I/O scan sweep impact is:

| I/O Scan Sweep Impact | = | I/O Scan Overhead | + | Series 90*70 I/O Scan | + | Genius I/O Scan |

The following table shows the I/O scan overhead in milliseconds for each CPU:

Table A-5. I/O Scan Overhead *

CPU Model	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
I/O scan overhead	.0293	.0321	.1348	.1382	.0638	.1756	.0638	0.38

   *   Times are in milliseconds. For information not available when this manual was being printed (represented by a dash: –), refer to the IPI for each CPU.

Note

I/O scan overhead impacts the sweep continuously.

Sweep Impact of Series 90-70 I/O Modules

The I/O scan of the Series 90-70 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
I/O Scan Overhead	Includes the setup for input and output scan and the selection of the main rack.
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 Series 90-70 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.

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 in table A-7.

Table A-6. Worksheet A: I/O Module Sweep Time

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	_____	
Seep 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 exander modules (same segment)—main rack	_____	
Sweep impact per analog input expander module (same segment)—main rack	x _____	= _____
Number of analog input exander 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 Series 90-70 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.

Note

Not all of the sweep time information was available at the time this manual was printed (the blank spaces in the table below). Refer to the IPI for the specific CPU for this information.

Table A-7. Sweep Impact Time for Model 70 I/O Modules and Racks *

	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
Rack Setup per expansion rack	.0010	.0010	.0010	.0010	.0010	.0010	.0010	.0500

Discrete I/O Modules
  per I/O module in main rack
  per I/O module in main rack w/point faults enabled
  per I/O module in expansion rack
  per I/O module in expansion rack w/point faults enabled
	.0074
	
	.0084
	.0119
	.0135
		.0086
		
		.0125
		.0144
		.0184
			.0209
			
			.0136
			.0226
			.0398
				.0105
				
				.0156
				.0110
				.0183
					.0144
					
					.0220
					.0154
					.0273
						.0090
						
						.0283
						.0102
						.0149
							.0144
							
							.0220
							.0154
							.0273
								.0800
								
								.1200
								.0900
								.1300

per fault message **	.1314	.1781	.4256	.1570	.3980	.1189	.3980	1.700
Rough Average per Point (no point faults)	-	-	-	-	-	-	-	3.1μs
Rough Average per Point (w/ point faults)	-	-	-	-	-	-	-	4.4μs

Table A-7. Sweep Impact Time for Model 70 I/O Modules and Racks – Continued

	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
Analog I/O Modules								
per input/output module in main rack	.0203	.0230	.0279	.0160	.0501	.0149	.0501	0.08
per input/output module w/point faults enabled in main rack	.0259	.0281	.0308	.0223	.0579	.0207	.0579	0.12
per input or output module in expansion rack	.0313	.0359	.0321	.0318	.0472	.0480	.0472	0.10
per input/output module w/point faults enabled in exp. Rack	.0451	.0499	.0464	.0523	.0688	.0761	.0688	0.15
per input expander module in same segment in main rack	.0166	.0183	.0152	.0169	.0187	.0188	.0187	0.02
per input expander module w/point faults enabled in same segment in main rack	.0249	.0271	.0242	.0259	.0279	.0275	.0279	0.03
per input expander module in new segment in main rack	.0189	.0199	.0203	.0227	.0267	.0366	.0267	0.03
per input expander module w/point faults enabled in new segment in main rack	.0272	.0308	.0279	.0324	.0395	.0514	.0395	0.05
per input expander module in same segment in expansion rack	.0537	.0534	.0558	.0567	.0277	.0570	.0277	0.06
per input expander module w/point faults enabled in same segment in expansion rack	.0796	.0822	.0824	.0836	.0558	.0856	.0558	0.10
per input expander module in new segment in expansion rack	.0557	.0566	.0598	.0653	.0356	.0705	.0356	0.10
per input expander module w/point faults enabled in new segment in expansion rack	.0851	.0861	.0883	.0934	.0675	.1047	.0675	0.15
per fault message **	.1613	.2100	.2597	.3296	.5362	.9362	.5362	2.10
Rough Average per Channel (no point faults)	-	-	-	-	-	-	-	10.8μs
Rough Average per Channel (w/ point faults)	-	-	-	-	-	-	-	14.0μs

*   Times are in milliseconds, except for those identified as microseconds.
**   Faults for discrete Series 90-70 I/O modules are always polled for by the PLC CPU. When one occurs, it is always logged during one of the I/O scan phases of the sweep. These faults are only polled when point faults are enabled.

Note

Functions in bold type in Table A-7 above impact the sweep continuously. All other functions impact the sweep only when invoked. Also, not all of the timing information needed for the above table was available at print time for this manual (the blank spaces).

Sweep Impact of Genius I/O and GBCs

The sweep impact of Genius I/O and Genius Bus Controllers (GBC) is similar to that of Series 90-70 I/O. There is an overhead for the I/O scan that should be counted only once between the Series

90-70 I/O scan and the Genius I/O scan. There is also a per Genius Bus Controller sweep impact, a per scan segment sweep impact, and a transfer time (per word) sweep impact for all I/O data.

The sweep impact per Genius Bus Controller 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 Genius Bus Controller is one) is placed in the system.

2. Sweep impact to poll each Genius Bus Controller for background messages (datagrams). This part is an impact for every Genius Bus Controller in the system.

Note

> Both the first and second parts of the Genius Bus Controller's sweep impact may be eliminated by closing the System 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 COMMREQs will stop working while the window is closed. Communications with PCM and LAN modules will also stop.

3. Sweep impact to scan the Genius Bus Controller. This impact results from the PLC CPU notifying the Genius Bus Controller that its new output data has been transferred and commanding the Genius Bus Controller to ready its input data, as well as informing the Genius Bus Controller that the PLC has finished another sweep and is still in RUN mode.

A scan segment for Genius I/O consists of Genius blocks on the same bus with consecutive reference addresses and consecutive bus addresses. 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.

Note

> Not all of the sweep time information was available at the time this manual was printed (the blank spaces in the table below). Refer to the IPI for the specific CPU for this information.

Table A-8. Sweep Impact Time of Genius I/O and GBCs *

	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
Genius Bus Controller								
open system communications window	.0583	.0631	.0432	.0616	.1625	.1890	.1625	.6000
per Genius Bus Controller polling for background messages	.0072	.0083	.0081	.0086	.0160	.0200	.0160	.1000
per Genius Bus Controller I/O Scan	.5658	.5619	.4237	.6476	.5215	.8110	.5215	.7000
First Genius Bus Controller **	-	-	-	-	-	-	-	2.4000
Subsequent Genius Bus Controllers	-	-	-	-	-	-	-	1.6000
Genius I/O Blocks								
per I/O block scan segment	.0070	.0102	.0270	.0300	.0314	.0444	.0314	.0300
per I/O block scan segment w/point faults enabled	.0259	.0266	.0540	.0600	.0584	.0725	.0584	.0900
per byte discrete I/O data in the main rack	.0451	.0460	.0015	.0017	.0475	.0031	.0475	.0032
per byte discrete I/O data in expansion racks	.0468	.0471	.0020	.0025	.0442	.0045	.0442	.0045
per word analog I/O data in the main rack	.0634	.0643	.0011	.0011	.0604	.0042	.0604	.0029
per word analog I/O data in expansion racks	.0654	.0651	.0040	.0057	.0623	.0131	.0623	.0077
Asynchronous Events Fault Message	-	-	-	-	-	-	-	2.00

 * Times are in milliseconds, except for those identified as microseconds.
** The extra time for the first GBC is the same time as shown in the next table for the first intelligent option module.
     This time should be counted only once.

Note

Functions in bold type in Table A-8 above impact the sweep continuously. All other functions impact the sweep only when invoked.

Note

Not all of the timing information needed for the above table was available at print time for this manual (the blank spaces).

Use the following form for predicting the sweep impact due to Genius I/O. The sweep impact times can be found in table A-8.

Table A-9. Worksheet B: Genius I/O Sweep Time

Open system communications window	_____	= _____
GBC I/O scan	_____	
GBC poll for background messages	+ _____	= _____
Number of GBCs	x _____	= _____
Input block scan segments—number of	_____	
Input block scan segments—sweep impact	x _____	= _____
Output block scan segments—number of	_____	
Output 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	_____	
Seep 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—exapansion racks	x _____	= _____
Predicted Genius I/O Scan Impact		_____

Sweep Impact of FIP I/O and FBCs

The sweep impact of FIP I/O and FIP Bus Controllers (FBC) is similar to that of Series 90-70 I/O. There is an overhead for the I/O scan that should be counted only once between the Series 90-70 I/O scan and the FBC I/O scan. There is also a per FIP Bus Controller sweep impact, a per scan segment sweep impact; and a transfer time (per word) sweep impact for all I/O data.

The sweep impact per FIP Bus Controller 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 FIP Bus Controller is one) is placed in the system.

2. Sweep impact to poll each FIP Bus Controller for background messages (datagrams). This part is an impact for every FIP Bus Controller in the system.

Note

Both the first and second parts of the FIP Bus Controller's sweep impact may be eliminated by closing the System 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 COMMREQs will stop working while the window is closed. Communications with PCM and LAN modules will also stop.

3. Sweep impact to scan the FIP Bus Controller. This impact results from the PLC CPU notifying the FIP Bus Controller that its new output data has been transferred and commanding the FIP Bus Controller to ready its input data, as well as informing the FIP Bus Controller that the PLC has finished another sweep and is still in RUN mode.

A scan segment for FIP I/O consists of FIP Transfer Variables (TVAs) on the same bus with consecutive reference addresses containing up to 256 bytes. The scan segment processing is the same for analog and discrete scan segments.

Note

Some of the sweep time information was not available at the time this manual was printed (the blank spaces in the table below).

Table A-10. Sweep Impact Time of FIP I/O and FBCs *

	CPX 935 **	CPX 928 **	924/ 925**	914/ 915**	CPX 782**	781/ 782 **	CPX 772**
FIP Bus Controller							
open system communications window	.0583	.0631	.0432	.0616	.1625	.1890	.1625
per FIP Bus Controller polling for background messages	.0072	.0083	.0081	.0086	.0160	.0200	.0160
per FIP Bus Controller I/O Scan	.0839	.1147	___	___	.0931	___	.0931
First FIP Bus Controller ***							
Subsequent FIP Bus Controllers							
FIP I/O Blocks							
per I/O block scan segment	.0061	.0132	.0114	.0116	.0161	.0025	.0161
per I/O block scan segment w/point faults enabled	.0148	.0176	.0119	.0153	.0323	.0670	.0323
per byte discrete I/O data in the main rack	.0018	.0018	.0019	.0024	.0019	.0025	.0019
per byte discrete I/O data in expansion racks	.0025	.0021	.0024	.0026	.0024	.0044	.0024
per word analog I/O data in the main rack	.0012	.0012	.0008	.0011	.0011	.0071	.0011
per word analog I/O data in expansion racks	.0024	.0024	.0043	.0032	.0024	.0098	.0024
Asynchronous Events	___	___	___	___	___	___	___
Fault Message							

   *   Times are in milliseconds, except for those identified as microseconds.

  **   These are typical scan impacts. It is possible to incur up to an additional 2.1 milliseconds per scan segment, but this is not typical. This additional scan impact can usually be avoided when using synchronous scan sets that include only 1 FBC.

***   The extra time for the first FBC is the same time as shown in the next table for the first intelligent option module. This time should be counted only once.

Note

Functions in bold type in Table A-8 above impact the sweep continuously. All other functions impact the sweep only when invoked.

Also please note that the blank lines represent information not available at this time.

Use the following form for predicting the sweep impact due to FIP I/O. The sweep impact times can be found in table A-10.

Table A-11. Worksheet B: FIP I/O Sweep Time

Open system communications window		_____	= _____
FBC I/O scan		_____	
FBC poll for background messages	+ _____		= _____
Number of FBCs	x _____		= _____
Input block scan segments—number of		_____	
Input block scan segments—sweep impact	x _____		= _____
Output block scan segments—number of		_____	
Output block scan segments—sweep impact	x _____		= _____
Bytes of discrete I/O data on FBCs—main rack		_____	
Sweep impact/bytes of discrete I/O data—main rack	x _____		= _____
Bytes of discrete I/O data on FBCs—expansion racks		_____	
Seep impact/bytes of discrete I/O data—expansion racks	x _____		= _____
Words of analog I/O data on FBCs—main rack		_____	
Sweep impact/word analog I/O data—main rack	x _____		= _____
Words of analog I/O data on FBCs—expansion racks		_____	
Sweep impact/word analog I/O data—exapansion racks	x _____		= _____
Predicted FIP 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 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. However, at some point during the operation of the PLC, all of the exchanges will be scheduled to transfer data during the same sweep. Therefore, 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 |

Each Ethernet Global Data exchange 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 PLC 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.

Exchange Overhead*

	CPU Models	
Exchange Type	924/925	914/915
Consumed	61	106
Produced	88	133

* Times are in microseconds.

Byte Transfer Time

This is the time required to transfer the data between the PLC CPU module and the Ethernet module. The times shown in the following table represent the time to transfer one data byte.

Byte Transfer Time*

	CPU Models	
Exchange Type	924/925	914/915
Consumed	2.4	2.4
Produced	1.8	1.8

* Times are in microseconds

Table A-12. Worksheet: 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 Ethernet GlobalData Sweep Impact		_____

Note

If the PLC is configured for Microcycle Sweep Mode, the EGD sweep impact time is allocated to the Logic Window. Therefore, the time available to execution of the user program is reduced by the time required by Ethernet Global Data exchanges.

Sweep Impact of Intelligent Option Modules

Intelligent option modules include Programmable Coprocessor Modules (PCM), Alphanumeric Display Coprocessor (ADC) Modules, Graphics Display Coprocessor (GDC) Modules, MAP LAN Modules, Ethernet LAN Modules, and Genius Bus Controllers being used for Genius LAN capabilities. The sweep impact for these intelligent option modules is highly variable. The opening of the System Communications Window and the polling of each module have relatively small impacts compared to the sweep impact of CPU memory read or write requests.

The following equations show how to calculate the fixed sweep of each module.

```
PCM         = Polling Sweep Impact + Clock Refresh Impact once every 1/2 sec.

ADC         = Polling Sweep Impact + Clock Refresh Impact once every 1/2 sec.

GDC         = Polling Sweep Impact + Clock Refresh Impact once every 1/2 sec.

MAP LAN     = Polling Sweep Impact + LAN I/O Scan Impact

EthernetLAN = Polling Sweep Impact + LAN I/O Scan Impact

GBC         = Polling Sweep Impact + GBC I/O Scan Impact

FBC         = Polling Sweep Impact + FBC I/O Scan Impact
```

The table below shows the fixed sweep impact times in milliseconds for intelligent option modules. It also contains sweep impact times for reading and writing the PLC's system memories (includes all memories except %P and %L, which are slightly slower). The read and write service requests have two boundary conditions that change the times and are, therefore, broken up into three sets of times to reflect these boundary conditions.

Table A-13.   Fixed Sweep Impact Times for Intelligent Option Modules *

Sweep Impact Item	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
Intelligent Option Modules								
First module (open comm window)	-	-	-	-	-	-	-	-
	-	-	-	-	-	-	-	0.60
Per module (polling)	-	-	-	-	-	-	-	0.10
LAN module I/O Scan	.0751	.0833	.0471	.0586	.1960	.0665	.1960	0.15
PCM, ADC, GDC clock refresh	.1630	.1434	.1386	.1455	.3651	.3248	.3651	0.60
PLC Memory Access from IOMs								
Read/write 1 to 3 words. *	.4940	.5586	.3990	.4450	1.252	1.206	1.252	3.00
Read/write 4 to 128 words.	.6320	.6964	.4830	.5790	1.420	1.393	1.420	3.10
Read/write each additional 128 words.	.2274	.2495	.6530	.8030	.4140	1.835	.4140	1.00

*   Times are in milliseconds. Not all of the timing information needed for the above table was available at print time for this manual (the blank spaces). Refer to the IPI for each CPU for this information.

**   Reads can only fit 3 words into the basic message.
   Writes can fit 4 words before a 256 byte text buffer is needed.

Note

Functions in bold type in the previous table impact the sweep continuously. All other functions impact the sweep only when invoked.

I/O Interrupt Performance and Sweep Impact

There are several important performance numbers for I/O interrupt blocks or I/O-Triggered programs. The sweep impact of an I/O interrupt invoking an empty program or block 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 maximum I/O interrupt rate reflects the limit of I/O interrupts invoking a minimal program or block at a sustained rate over time. The time to execute the logic contained in the interrupt program or block will affect the limit by causing the PLC 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 or C code is executed in the I/O interrupt program or 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 program or 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 for the model 731 CPU. 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 Genius Bus Controller or a FIP Bus Controller). 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 unbound events.

Table A-14. I/O Interrupt Block Performance and Sweep Impact Times *

Sweep Impact Item	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
I/O interrupt sweep impact	-	-	-	-	-	-	-	0.95
I/O interrupt response time								
Input card filter time	-	-	-	-	-	-	-	0.30
+ typical interrupt latency	-	-	-	-	-	-	-	1.40
+ interrupt to logic time	-	-	-	-	-	-	-	0.48
Minimum response time	.3851	.3851	.313		.5402	.675	.5402	0.78
Typical response time	.3873	.3873	.316		.5421	.679	.5421	2.18
Maximum response time	.5424	.5424	.627		1.0528	1.633	1.0528	3.60
I/O interrupt rate limit								450 ints/sec
I/O interrupt rate limit w/no IOMs								750 ints/sec

*   Times are in milliseconds. Not all of the timing information needed for the above table was available at print time for this manual (the blank spaces).

Table A-15. I/O-Triggered Interrupt Performance and Sweep Impact Times *

Sweep Impact Item	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772
I/O interrupt sweep impact							
I/O interrupt response time							
Input card filter time							
+ typical interrupt latency							
+ interrupt to logic time							
Minimum response time	.3046	.3259			.4796		.4796
Typical response time	.3072	.3273			.4797		.4797
Maximum response time	.4618	.4822			.9921		.9921
I/O interrupt rate limit							
I/O interrupt rate limit w/no IOMs							

*   Times are in milliseconds. Not all of the timing information needed for the above table was available at print time for this manual (the blank spaces).

The following form is a worksheet for the sweep impact times of programmer sweep impact, intelligent option modules, and I/O Interrupts. (Refer to tables A-4, A-10, and A-11.)

Note

Not all of the timing information needed for the above table was available at printing time for this manual (the blank spaces). Refer to the IPI for each CPU for this information.

Table A-16. Worksheet C: Programmer, IOM, I/O Interrupt Sweep Time

Programmer sweep impact		= _____
IOM—first module (open comm window)	_____	
IOM—per module (polling)	+ _____	
LAN module I/O scan	+ _____	
Total IOM Sweep Impact		= _____
PLC 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, typical, and maximum interrupt response times reflect the time from when a single timed interrupt occurs until the first line of ladder logic or C code is executed in the timed interrupt program or 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 the CPU's maximum latency time. This interrupt latency time is valid, except when one of the following operations occurs:

- The programmer is attached

- A store of logic, ALT-S store, RUN mode store, or word-for-word change occurs

- A fault condition (logging of a fault) occurs

- Another timed interrupt or I/O interrupt occurs

Any one of these events extends the interrupt period beyond the typical value. However, the latency of an interrupt occurring during the processing of a preceding timed or I/O interrupt is unbounded. For interrupts, the worst case is that every timed and 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 unbound events.

Table A-17. Timed Interrupt Performance and Sweep Impact Times *

Sweep Impact Item	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772	731/732 771/772
Timed interrupt sweep impact	-	-	-	-	-	-	-	1.15

Timed interrupt response time
  Typical interrupt latency
  + interrupt to logic time
	-
	-
		-
		-
			-
			-
				-
				-
					-
					-
						-
						-
							-
							-
								1.40
								0.68

Minimum response time Typical response time Maximum response time	- - -	- - -	.108 .143 .163	.147 .185 .219	 - -	.227 .346 .464	- - -	0.68 2.08 3.50

*   Times are in milliseconds.   Not all of the timing information needed for the above table was available at print time for this manual (the blank spaces).

Table A-18. I/O-Triggered Interrupt Performance and Sweep Impact Times

Sweep Impact Item	CPX 935	CPX 928	924/ 925	914/ 915	CPX 782	781/782 788/789	CPX 772
Timed interrupt sweep impact	-	-	-	-	-	-	-

Timed interrupt response time
    Typical interrupt latency
    + interrupt to logic time
	-
	-
		-
		-
			-
			-
				-
				-
					-
					-
						-
						-
							-
							-

Minimum response time Typical response time Maximum response time	- - -	- - -	- - -	- - -	- - -	- - -	- - -

*   Times are in milliseconds.   Not all of the timing information needed for the above table was available at
      print time for this manual (the blank spaces).

Examples of Calculating Predicted Sweep Times

The following two examples are provided to show how to calculate predicted sweep times. The
first example is a small system and the second is a large system. Neither of these sweep time
estimates include a time for logic execution. In both of these systems, the calculated sweep is for
normal sweep time with point faults disabled, the PCM idle, and the programmer not attached. The
times used in the calculation are extracted from tables A-3, A-5, A-7, A-8, and A-10. Sample
forms for calculating predicted sweep times are provided after the examples.

Small System

PS	CPU 731	BTM	32PT Input	32PT Input	32PT Output	32PT Output	8CHN Analog Input	4CHN Analog Output	PCM
0	1	2	3	4	5	6	7	8	9

MAIN RACK

Sweep Calculations

| Predicted Sweep | = | Base Sweep | + | I/O Scan | + | PCM Impact |

Base Sweep Time	=	2.00

I/O Scan Impact = I/O Scan Overhead + Series 90-70 I/O Scan Impact

Number of discrete I/O modules—main rack	4		
Sweep impact time per discrete I/O module	x 0.08	=	0.32
Number of analog base and output modules—main rack	2		
Sweep impact time per analog base and output module	x 0.08	=	0.16
I/O Scan Impact = 0.32 + 16	.48	=	0.48
I/O scan overhead	.38	=	0.38

PCM Impact

First module (open comm window)	.60		
Per module (polling)	+ .10	=	0.70
Predicted Sweep Time		=	4.04

Note

Times are in milliseconds.

Large System

PS	CPU 781	BTM	GBC 20Blks	GBC 20Blks	GBC 16Blks	GBC 16Blks	PCM	PCM	ENET LAN
0	1	2	3	4	5	6	7	8	9

MAIN RACK

PS	BRM	32PT Input	32PT Input	32PT Output	32PT Output	8CHN Analog Input	16CHN Analog Expndr	4CHN Analog Output	4CHN Analog Output
0	1	2	3	4	5	6	7	8	9

RACK1

PS	BRM	32PT Input	32PT Input	32PT Output	32PT Output	8CHN Analog Input	16CHN Analog Expndr	4CHN Analog Output	4CHN Analog Output
0	1	2	3	4	5	6	7	8	9

RACK2

Note

The Genius block configuration used for this example is ala 16-point grouped (QI) blocks with all bus addresses having contiguous reference addresses.

Predicted Sweep Calculations

Predicted Sweep = Base Sweep + IOM Impact

Base Sweep Time	0.10	=	0.10
I/O Scan overhead	0.20		
90-70 I/O scan impact (see table _____, Worksheet A)	+ 0.80		
Genius I/O scan impact (see table _____, Worksheet B)	+ 5.88		
I/O scan overhead		=	6.88
PCM impact	x .30		
Number of PCMs	2		
LAN impact	0.60		
IOM impact	+ .08	=	.68
Predicted Sweep Time		=	8.66

Note

Times are in milliseconds.

Table A-19. Worksheet A

Number of expansion racks			2	
Sweep impact per expansion rack	x	.03	=	.06
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			8	
Seep impact per discrete I/O module—expansion rack	x	.05	=	.40
Number of analog input base and output modules—main rack			6	
Sweep impact per analog input base and output module—main rack	x	.04	=	.24
Number of analog input exander modules (same segment)—main rack			_____	
Sweep impact per analog input expander module (same segment)—main rack	x	_____	=	_____
Number of analog input exander 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			2	
Sweep impact per analog input base and output module (same seg.)—exp. rack	x	.05	=	.10
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 Series 90-70 I/O Module Sweep Impact				.80

Table A-20.   Worksheet B

Open system communications window			0.30	= 0.30
GBC I/O scan			0.94	
GBC poll for background messages	x		0.04	= 0.98
Number of GBCs	x		4	= 3.92
Input block scan segments—number of			4	
Input block scan segments—sweep impact	x		0.02	= 0.08
Output block scan segments—number of			4	
Output block scan segments—sweep impact	x		0.02	= 0.08
Bytes of discrete I/O data on GBCs—main rack			288	
Sweep impact/bytes of discrete I/O data—main rack	x		.0018	= .5184
Bytes of discrete I/O data on GBCs—expansion racks			_____	
Seep 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—exapansion racks	x	_____		= _____
Predicted Genius I/O Scan Impact				5.88

Table A-21. Sample Worksheet A

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	_____	
Seep 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 exander modules (same segment)—main rack	_____	
Sweep impact per analog input expander module (same segment)—main rack	x _____	= _____
Number of analog input exander 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 Series 90-70 I/O Module Sweep Impact		_____

Table A-22. Sample Worksheet B

Programmer sweep impact		= _____
IOM—first module (open comm window)	_____	
IOM—per module (polling)	+ _____	
LAN module I/O scan	+ _____	
Total IOM Sweep Impact		= _____
PLC memory access from IOMs		= _____
I/O interrupt sweep impact	_____	
I/O interrupt response time	+ _____	= _____
Predicted Sweep Time (Other)		_____

Relative CPU Performance Comparison

This section contains a relative CPU performance comparison based on lab test results on the Series 90-70 CPUs.

Test Program

The test program used for the performance tests consisted of a 190 Kilobyte folder run on a CPX935 CPU. Sweep time was approximately 115 ms. In this test program, the Main block calls the same subroutine (S1) 4 times. The S1 subroutine, in turn, calls 60 other subroutines (S2 – S61) which each contain the same logic. The logic consists of :

- 140 contacts
- 50-70 coils
- 5 ADDs
- 2 DNCTRs
- 5 UPCTRs
- 10 EQ_INT
- 3 FIFO RDs
- 12 MOVEs
- 1 MUL_INT
- 1 NE_INT
- 8 ONDTRS
- 1 CALL
- 2 Bit Shifts
- 4 SUB_INTs
- 4 TMRs
- 1 DIVIDE
- 1 PI-ISA

Interpreting the Chart

The CPX935 performance time for running the test program was assigned a value of 100%, and the other CPUs were compared to that standard. The chart shows that the CPX935 had the fastest time. The CPU782 took the longest, 446% longer than the CPX935.

Note

The figures in the following chart were measured for the test program described above. They will not be exactly the same for different programs, but should provide a basic guide to relative CPU performance.

Relative Time Required to Run the Test Program
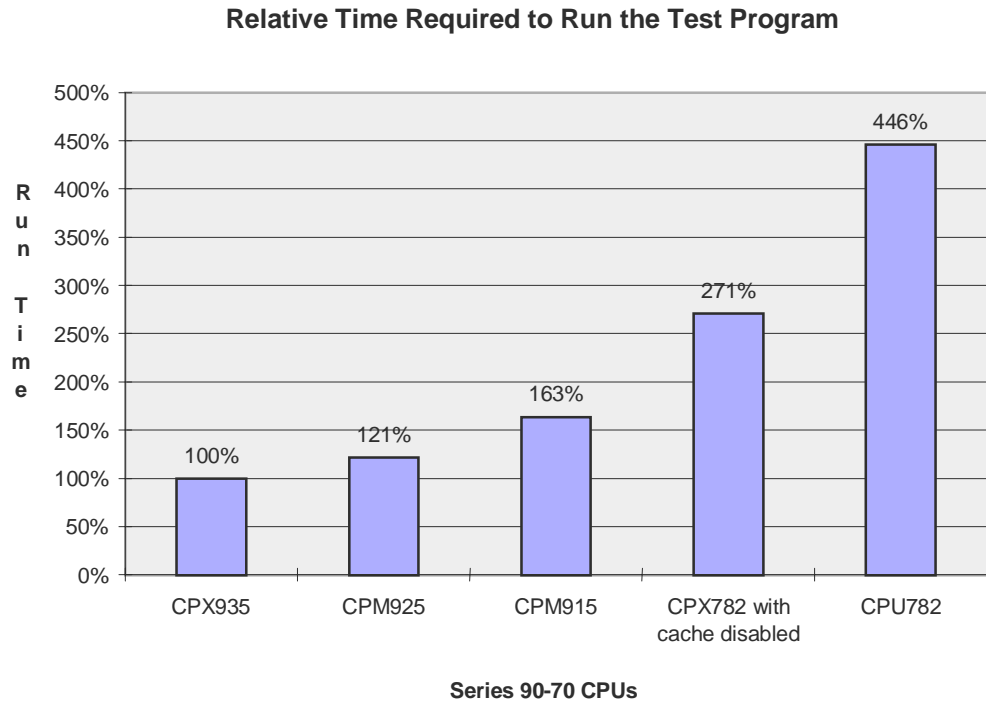
Figure A-1. Chart of Relative CPU Performance

Interpreting Faults Using Logicmaster 90-70 Software

The Series 90-70 PLC maintains two fault tables, the I/O fault table for faults generated by I/O devices (including I/O controllers) and the PLC fault table for internal PLC faults. The information in this appendix will enable you to interpret the message structure format when reading these fault tables.

Both tables contain similar information.

- The PLC fault table contains:

  □ Fault location.

  □ Fault description.

  □ Date and time of fault.

- The I/O fault table contains:

  □ Fault location.

  □ Circuit number.

  □ Reference address.

  □ Fault category.

  □ Fault type.

  □ Date and time of fault.

The Series 90-70 PLC maintains additional information on each fault that is helpful when troubleshooting your system. This information is called CTRL-F data, or Fault Detail. You can access this information by highlighting the fault and pressing the **CTRL** and **F** keys together.

PLC Fault Table

The following diagram identifies each field in the fault entry:

```
00    000080   00030100   0B02   0400   FF16010400000000
```

- Fault Extra Data
- Error Code
- Fault Action
- Fault Group
- Task
- Slot
- Rack
- Spare
- Long/Short
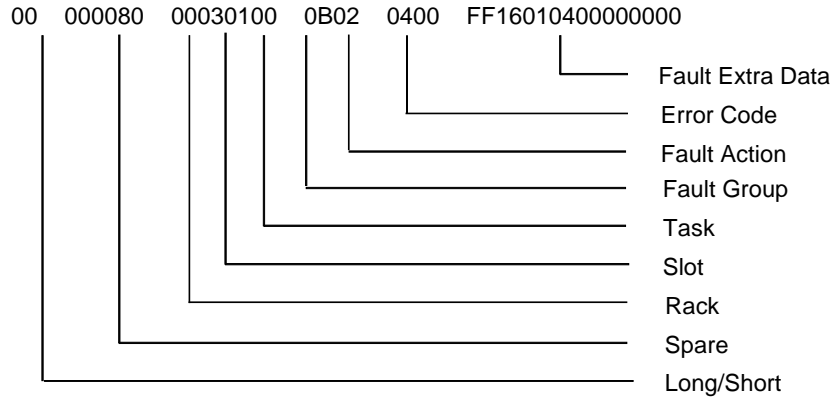
The following paragraphs describe each field in the fault entry. Included are tables describing the range of values each field may have.

Long/Short Indicator

This byte indicates whether the fault contains 8 bytes or 24 bytes of fault extra data.

Type	Code	Fault Extra Data
Short	00	8 bytes
Long	01	24 bytes

Spare

These 3 bytes are pad bytes, used to make the PLC fault table entry exactly the same length as the I/O fault table entry.

Rack

The rack number ranges from 0 to 7. Zero is the main rack, containing the PLC. Racks 1 through 7 are expansion racks, connected to the PLC through a Bus Transmitter Module in the main rack and Bus Receiver Modules in the expansion racks.

Slot

The slot number ranges from 0 to 9. The PLC CPU always occupies slot 1 in the main rack (rack 0).

Task

The task number ranges from 0 to +65,535. Sometimes the task number provides additional information to PLC engineers; typically, however, the task number can be ignored.

PLC Fault Group

Fault 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.

Table B-1 lists the possible fault groups in the PLC fault table. Group numbers less than 80 (Hex) are maskable faults, while group numbers greater than or equal to 80 (Hex) are non-maskable faults.

The last non-maskable fault group, Additional PLC Fault Codes, is declared for the handling of new fault conditions in the system without the PLC having to specifically know the alarm codes. All unrecognized PLC-type alarm codes belong to this group.

Table B-1. PLC Fault Groups

Group Number		Group Name	Fault Action
Decimal	**Hexa-decimal**		
1	1	Loss of, or missing, rack	Fatal
4	4	Loss of, or missing, option module	Diagnostic
5	5	Addition of, or extra, rack	Diagnostic
8	8	Addition of, or extra, option module	Diagnostic
11	B	System configuration mismatch	Fatal
12	C	System bus error	Diagnostic
13	D	PLC CPU hardware failure *	Fatal
14	E	Non-fatal module hardware failure	Diagnostic
16	10	Option module software failure	Diagnostic
17	11	Program block checksum failure	Fatal
18	12	Low battery signal	Diagnostic
19	13	Constant sweep time exceeded	Diagnostic
20	14	PLC system fault table full	Diagnostic
21	15	I/O fault table full	Diagnostic
22	16	User Application fault	Diagnostic
–	–	Additional PLC fault codes	As specified
128	80	System bus failure	Fatal
129	81	No user's program on power-up	Informational
130	82	Corrupted user RAM detected	Fatal
131	83	Window completion failure in Constant Sweep mode (that is, all windows failed to receive their allotted time)	Informational
132	84	Password access failure	Informational
134	86	Null system configuration for RUN mode	Informational
135	87	PLC CPU software failure	Fatal
136	88	More than the allowable number of I/O bus controllers were found in the system	Fatal
137	89	PLC sequence-store failure	Fatal

* The PLC OK LED will flash on and off to indicate that the failure was not serious enough to prevent programmer communications from retrieving the fault table information.

PLC Fault Action

Each fault may have one of three actions associated with it.

Table B-2.   PLC Fault Actions

Fault Action	Action Taken by CPU	Code
Informational	Log fault in fault table.	1
Diagnostic	Log fault in fault table. Set fault references.	2
Fatal	Log fault in fault table. Set fault references. Go to Stop mode.	3

Error Code

The error code further describes the fault. Each fault group has its own set of error codes.

The first table below shows error codes for the PLC Software Error Group (Group 87H).

Table B-3. Alarm Error Codes for PLC CPU Software Faults

Decimal	Hexa-decimal	Name
20	14	Corrupted PLC Program Memory.
39	27	Corrupted PLC Program Memory.
82	52	Backplane Communications Failed.
90	5A	PLC Stopped by Service Request # 13.
123	7B	Remote I/O Scanner Communications Failure.
149	95	Store from Flash on Power-Up Failed. **Note:** The first byte of the Fault Extra Data describes why the store from flash failed:

		Error	**Fault Extra Data Value**	**Description**
		DEVICE_NOT_AVAILABLE	CF	Specific device is not available in the system.
		BAD_DEVICE_DATA	CC	Data stored on device has been corrupted and is no longer reliable. Or, Flash Memory has not been initialized.
		DEVICE_RW_ERROR	CB	Error occurred during a read/write of the Flash Memory device.
		FLASH_INCOMPAT_ERROR	8E	Data in Flash Memory is incompatible with the PLC CPU firmware release due to the CPU firmware revision numbers, the instruction groups supported, or the CPU model number.
		ITEM_NOT_FOUND_ERROR	8D	One or more specified items were not found in Flash Memory.

Decimal	Hexa-decimal	Name
All	All	PLC CPU Internal System Error

The next table shows the error codes for all the other fault groups.

Table B-4. Alarm Error Codes for PLC CPU Faults

Decimal	Hexa-decimal	Name
		PLC Error Codes for Loss of Option Module Group
3	3	Bus Transmitter Module Found in Expansion Rack
22	16	Analog Expander Located to Left of Base Converter Module
25	19	Lost Analog Expander Module
44	2C	Option Module Soft Reset Failed
45	2D	Option Module Soft Reset Failed
59	3B	Loss of, or missing communications driver
60	3C	Module in firmware update mode
65	41	Module is in Standalone mode; mail system not initialized
75	4B	CFG ESCM Not Compatible—Upgrade Ports 1 and 2 to work with CPU
76	4C	CFG 486 Not Compatible—Upgrade firmware to work with Ports 1 and 2
81	51	ESCM Reset Req—Port 1 or 2 requested a reset. Cycle power when Port 1 or 2 completes the update.
255	FF	Option Module Communication Failed
		Error Codes for Addition of, or Extra Rack Group
1	1	Addition of, or Extra Rack
		Error Codes for Reset of, Addition of, or Extra Option Module Group
2	2	Module Restart Complete
3	3	LAN Interface Restart Complete; Running a Utility
All others		Reset of, Addition of, or Extra Option Module
		Error Codes for Module Hardware Error Group
1	1	Non-fatal LAN Hardware error
416	1A0	Required 12V PS failed or missing
450	1C2	LAN Controller Underrun/Overrun; Resuming
451	1C3	LAN Interface Failure; Switched Off Network
452	1C4	LAN Network Problem; Performance Degraded
453	1C5	LAN Severe Network Problem; Attempting Recovery
454	1C6	LAN Transceiver Fault; Off Network Until Fixed
		Error Codes for Option Module Software Failure Group
1	1	Unsupported Board Type
2	2	COMREQ _ mailbox full on outgoing message that starts the COMREQ
3	3	COMREQ _ mailbox full on response
4	4	More Than One BTM in Rack
5	5	Backplane Communications with PLC; Lost Request
10	A	Error with LAN interaction
11	B	Resource (alloc, tbl ovrflw, etc.) error
12	C	VME backplane error
13	D	User program error
401	191	Module Software Corrupted; Requesting Reload
402	192	LAN System Software Fault; Resuming
403	193	LAN System Software Fault; Aborted Assoc and Resuming
404	194	LAN System Software Fault; Restarted LAN I/F
405	195	LAN System Software Fault; Reinitializing LLC

Table B-4. Alarm Error Codes for PLC CPU Faults - Continued

Decimal	Hexa-decimal	Name
		Error Codes for System Configuration Mismatch Group
2	2	Genius Block Model Number Mismatch
4	4	Genius Block I/O Type Mismatch
7	7	Daughter Board Mismatch
8	8	Analog Expansion Mismatch
9	9	Genius Block Broadcast Control Data (BCD) Length Mismatch
10	A	Unsupported Feature
11	B	Revision A BTM not in Right Slot
14	E	LAN Duplicate MAC Address
15	F	LAN Duplicate MAC Address Resolved
16	10	LAN MAC Address Mismatch
17	11	LAN Soft Switch/Modem Mismatch
19	13	Genius Block Direct Control Data (DCD) Length Mismatch
23	17	Program exceeds memory limits
29	1D	Incompatible scheduling mode
30	1E	Reference length mismatch
31	1F	Invalid configuration parameters
32	20	New configuration requires reset
36	24	I/O specification mismatch
37	25	Controller reference out of range
39	27	Bad interrupt trigger
		Error Codes for System Bus Error Group
4	4	Unrecognized VME Interrupt Error
All others		System Bus Error
		Error Codes for Program Block Checksum Group
0	0	Corruption of program block header in the Series 90-70 PLC
1	1	Corruption of stored OMF records stored in Series 90-70 PLC
2	2	Corruption of stored OMF records stored in Series 90-70 PLC
3	3	Program or program block checksum failure
		Error Codes for Low Battery Signal
0	0	Failed battery on PLC CPU or other module
1	1	Low battery on PLC CPU or other module
2	2	Failed battery from VME backplane
3	3	Low battery from VME backplane

Table B-4. Alarm Error Codes for PLC CPU Faults - Continued

Decimal	Hexa-decimal	Name
Error Codes for User Application Fault Group		
1	1	Indirect Reference Address Out of Range
2	2	PLC Watchdog Timer Timed Out
3	3	GBC COMREQ
4	4	GBC Bkgnd msg – Bad Genius Bus Request
5	5	COMREQ – WAIT mode not available for this command
6	6	COMREQ – Bad Task ID
7	7	Application Stack Overflow
8	8	LAN Data Memory Exhausted – Check Parms; Resuming
9	9	Bad Remote Application Request; Discarded Request
10	A	Bad Local Application Request; Discarded Request
11	B	LAN I/F Capacity Exceeded; Discarded Request
12	C	LAN PROM/Software Mismatch; Running Soft Sw Util
13	D	LAN I/F Can't Init – Check Parms; Running Soft Sw Util
14	E	Run-time error detected in an external block
15	F	SORT function in an interrupt did not execute
17	11	Standalone Run-Time Error
28	1C	Program Exceeded Wind Down
29	1D	Program Not Readied
Error Codes for System Bus Failure Group		
1	1	Operating system
Error Codes for Corrupted User RAM on Powerup Group		
1	1	Corrupted User RAM on Power-up
2	2	Illegal Boolean Opcode Detected
5	5	Partial Store failure on second pass of parsing OMF
6	6	Corrupted Remote I/O Scanner EEPROM; Config Lost
Error Codes for PLC CPU Hardware Faults		
2169	879	Remote I/O Scanner Comm Failure; Verify Bus
2172	87C	Remote I/O Scanner Serial Bus Address Conflict
2048	800	Remote I/O Scanner Hardware Fault
to	to	
4095	FFF	
All other codes		PLC CPU Hardware Failure

PLC Fault Extra Data

The Fault Extra Data field in the PLC fault table contains details of the fault entry. Some examples of what data may be present are:

System Configuration Mismatch

The following error codes in the System Configuration Mismatch group supply fault extra data:

Table B-5. PLC Fault Extra Data – System Configuration Mismatch

Fault Extra Data Byte Number	Model Number Mismatch (2 Decimal)
[0]	FF
[1]	Bus address
[2]	Installed module's model number
[3]	Configured model number
Fault Extra Data Byte Number	**I/O Type Mismatch (4 Decimal)**
[0]	FF
[1]	Bus address
[2]	Installed module's I/O type
[3]	Configured module's I/O type
Fault Extra Data Byte Number	**BCD Length Mismatch (9 Decimal)**
[0]	FF
[1]	Bus address
[2]	Module's broadcast data length
[3]	Configured module's broadcast data length
Fault Extra Data Byte Number	**Unsupported Feature (10 Decimal)**
[0]	0A – Unsupported ESCM Configuration
[1]	01 – Unsupported baud rate
	02 – Unsupported protocol
[2]	00 - Port 1 (RS-232 port)
	01 - Port 2 (RS-485 port)
Fault Extra Data Byte Number	**Unsupported Feature (10 Decimal)**
[0}	45 - ASCII 'E' *
[1]	53 - ASCII 'S'
[2]	43 - ASCII 'C'
[3]	4D - ASCII 'M'
[4]	50 - ASCII 'P'
[5]	31 - ASCII '1' Port 1 (RS-232 Port)
	32 - ASCII '1' Port 2 (RS-485 Port)
Fault Extra Data Byte Number	**DCD Length Mismatch (19 Decimal)**
[0]	FF
[1]	Bus address
[2]	Module's directed data length
[3]	Configured module's directed data length

*Indicates that Port 1 or Port 2 is enabled, but the CPU does not support Embedded Serial COM Module ports. The ASCII value in the sixth byte (offset 5) of the extra data indicates which port is enabled in the stored configuration.

The following table shows the Genius numbers used when a model number mismatch occurs.

Table B-6. Genius Block Model Numbers

Number		Description
Decimal	**Hexadecimal**	**Description**
4	4	Genius Network Interface (GENI)
5	5	Phase B Hand Held Monitor
6	6	Phase B Series Six Genius Bus Controller with Diagnostics
7	7	Phase B Series Six Genius Bus Controller 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
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

If the model number is 7FH (Genius Network Adapter), the block may be one of the following. (The GENA Application ID is shown for reference.)

Table B-7. GENA Application ID Numbers

Number		Description
Decimal	**Hexadecimal**	
131	83	115Vac/230Vac/125Vdc Power Monitor Module
132	84	24/48Vdc Power Monitor Module
160	A0	Genius Remote 90-70 Rack Controller

When the system configuration mismatch is an I/O type mismatch, the installed module I/O type is one of the following:

Table B-8. Genius Installed Module I/O Types

Value	Description
01	Input only
02	Output only
03	Combination

When the system configuration mismatch is an I/O type mismatch, the configured module I/O type is one of the following. (All values are in hexadecimal.)

Table B-9. Genius Configured Module I/O Types

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

Program Block Checksum Failure

The name of the offending program block is contained in the first eight bytes of the Fault Specific Data field.

PLC CPU Hardware Failure (RAM Failure)

For a RAM failure in the PLC CPU (one of the faults reported as a PLC CPU hardware failure), the address of the failure is stored in the first four bytes of the field.

Application Fault

Indirect Reference Overflow: The offset address of where the call was made is located in the first two bytes. The offending reference (segment selector and offset) is located in the next four bytes. The name of the program block in which the function call resides is contained in the next eight bytes.

Bad COMREQ Status Pointer: The first byte contains a hex FF. The next four bytes contain the segment selector and offset of the status pointer into which the PLC CPU could not write.

Bad Genius Bus Request: Four bytes are used, unless the request is a read or write device. In these two datagrams, eight bytes are used.

Table B-10. Fault Specific Data - Bad Genius Bus Request

Fault Specific Data	Bad Genius Bus Request
[0]	FF
[1]	Bus address of requestor
[2]	Function code
[3]	Subfunction code
[4]	Segment selector, if Read/Write device
[5]	LSB of offset, if Read/Write device
[6]	MSB of offset, if Read/Write device
[7]	Length, if Read/Write device

PLC Fault Time Stamp

The six-byte time stamp is the value of the system clock when the fault was recorded by the PLC CPU. (Values are coded in BCD format.)

Table B-11. PLC Fault Time Stamp

Byte Number	Description
1	Seconds
2	Minutes
3	Hours
4	Day of the month
5	Month
6	Year

Genius Block I/O Type Mismatch Example

The Genius Block I/O Type Mismatch fault entry is explained below. (All data is in hexadecimal.)

Field	Value	Description
Long/Short	00	This fault contains 8 bytes of fault extra data
Rack	00	Main rack (rack 0)
Slot	03	Slot 3. In this configuration, slot 3 contains a Genius Bus Controller
Task	01	Single channel bus controller has only one task
Fault Group	0B	System Configuration Mismatch fault
Fault Action	02	Diagnostic fault
Error Code	04	Error code 04 in the System Configuration Mismatch group is a Genius Block I/O Type Mismatch
Fault Extra Data		IO Type Mismatch error has four bytes of fault extra data
[0]	FF	Flag byte
[1]	16	Serial bus address: 16 hex is 22 decimal
[2]	01	Installed module type. 01 is an inputs only module
[3]	04	Configured module I/O type. 04 is a combination module

The configuration file stored from the programming software shows that the device at serial bus address 22 is a combination (mixed) module. However, the "Read ID Reply" message the Genius Bus Controller received from the device at serial bus address 22 shows that the device is configured for inputs only. The Genius Bus Controller logged this fault when it detected the mismatch.

```
00    000080   00030100   0B02   0400   FF16010400000000
 |       |         |        |      |            |
 |       |         |        |      |            |___ Fault Extra Data
 |       |         |        |      |_____ Error Code
 |       |         |        |_____ Fault Action
 |       |         |_____ Fault Group
 |       |     |_____ Task
 |       |  |_____ Slot
 |       |_____ Rack
 |_____ Spare
 |_____ Long/Short
```
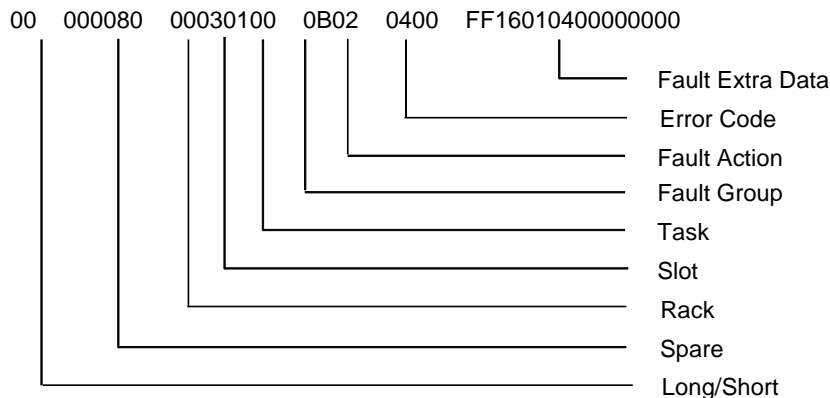
Low Battery Signal Example

The Low Battery fault is described below. (All data is in hexadecimal.)

Field	Value	Description
Long/Short	00	This fault contains 8 bytes of fault extra data.
Rack	00	Main rack (rack 0).
Slot	01	Slot 1. In all configurations, slot 1 in rack 0 contains the PLC CPU.
Task	00	PLC CPU.
Fault Group	12	Low battery signal.
Fault Action	02	Diagnostic fault.
Error Code	01	Error code 01 in the Low Battery group is one of four low battery conditions detected by the PLC CPU.
Fault Extra Data		No fault extra data for a low battery signal.

This fault occurred because the PLC CPU detected a low battery signal.

```
00  1F4100  00010000  1202  0100  01010B300BE069602
```
                                            Fault Extra Data
                                     Error Code
                                 Fault Action
                             Fault Group
                         Task
                     Slot
                 Rack
             Spare
         Long/Short
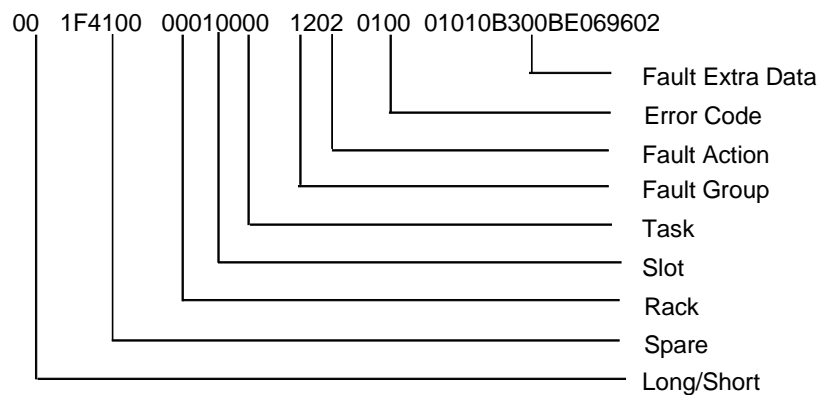
User Application Fault Example

The User Application Fault is described below. (All data is in hexadecimal.)

Field	Value	Description
Long/Short	00	This fault contains 8 bytes of fault extra data
Rack	00	Main rack (rack 0)
Slot	03	Slot 3. In this configuration, slot 3 contains a Genius Bus Controller
Task	7F	When the GBC registers a User Application fault in the PLC fault table, it places a 7F in the task byte of the fault.
Fault Group	16	User Application fault
Fault Action	01	Informational fault
Error Code	04	Error code 04 in the Application Fault group is a Bad Genius Bus Request. This fault occurs when the Genius Bus Controller receives a Read or Write Device datagram from another device on the Genius Bus that cannot be successfully completed.
Fault Extra Data		Bad Genius Bus Request error has eight bytes of fault extra data
[0]	FF	Flag byte
[1]	1D	Serial bus address: 1D hex is 29 decimal
[2]	20	Function code in the datagram: a GE Fanuc datagram
[3]	1E	Subfunction code in the datagram: a Read Device
[4]	08	Segment selector: 08 is %R memory
[5]	FF	Least significant byte of offset
[6]	FF	Most significant byte of offset
[7]	03	Length of data to read: 3 words

The Genius Bus Controller received a Read Device datagram from serial bus address 29, which requested three words of %R memory be read starting at %R65536. Since this is beyond the range of the largest value %R can have, the bus controller registered an informational user application fault in the PLC fault table.

```
00  F28407  00037F00   1601  0400  FF1D201E08FFFF03
```

Fault Extra Data
Error Code
Fault Action
Fault Group
Task
Slot
Rack
Spare
Long/Short
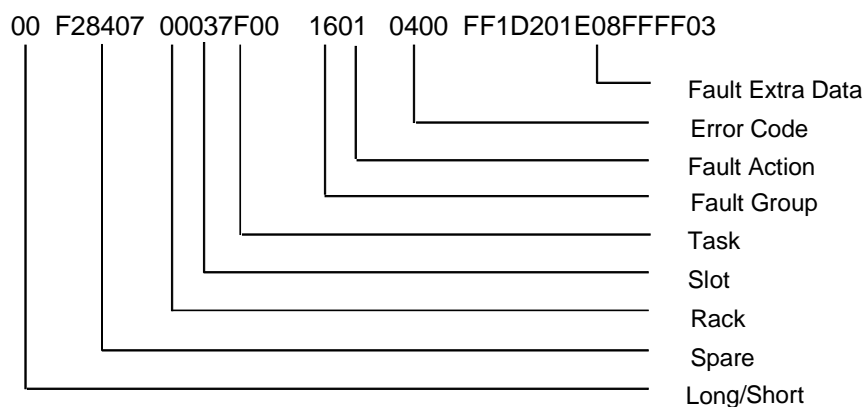
I/O Fault Table

The following diagram identifies the hexadecimal information displayed in each field in the fault entry.

```
02  1F0100  00030101FF7F  0302  02  00  00  84000000000003
```

Fault Specific Data
Fault Description
Fault Type
Fault Category
Fault Action
Fault Group
Point
Block
I/O Bus
Slot
Rack
Reference Address
Long/Short
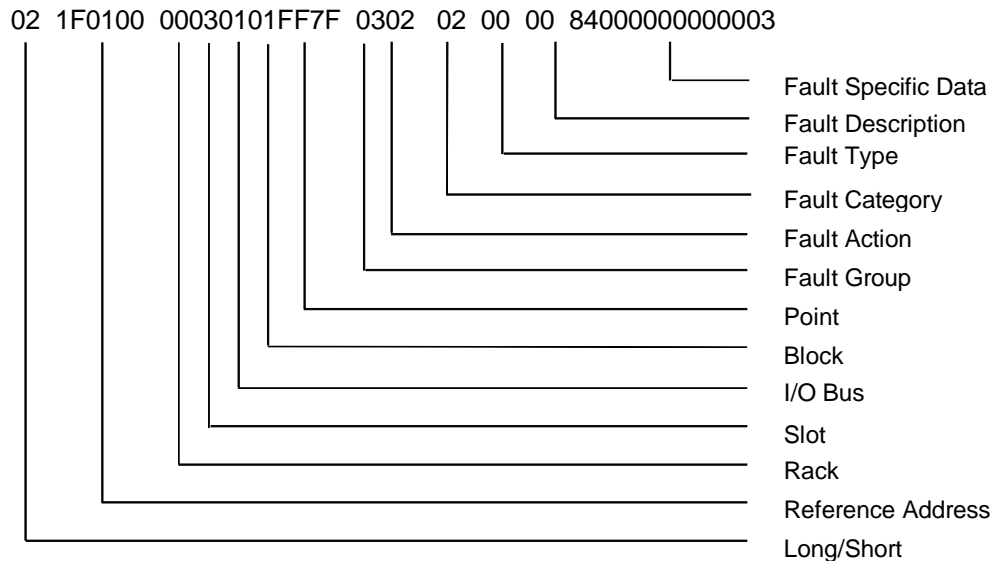
The following paragraphs describe each field in the I/O fault table. Included are tables describing the range of values each field may have.

Long/Short Indicator

This byte indicates whether the fault contains 5 bytes or 21 bytes of fault specific data.

Table B-12. I/O Fault Table Format Indicator Byte

Type	Code	Fault Specific Data
Short	02	5 bytes
Long	03	21 bytes

Reference Address

Reference address is a three-byte address containing the I/O memory type and location (or offset) in that memory which corresponds to the point experiencing the fault. Or, when a Genius block fault or integral analog module fault occurs, the reference address refers to the first point on the block where the fault occurred.

Table B-13. I/O Reference Address

Byte	Description	Range
0	Memory Type	0 - FF
1-2	Offset	0 - 12K (decimal)

The memory type byte is one of the following values.

Table B-14. I/O Reference Address Memory Type

Name	Value (Hexadecimal)
Analog input	0A
Analog output	0C
Analog grouped	0D
Discrete input	10 or 46
Discrete output	12 or 48
Discrete grouped	1F

I/O Fault Address

The I/O fault address is a six-byte address containing rack, slot, bus, block, and point address of the I/O point which generated the fault. The point address is a word; all other addresses are one byte each. All five values may not be present in a fault.

When an I/O fault address does not contain all five addresses, a 7F hex appears in the address to indicate where the significance stops. For example, if 7F appears in the bus byte, then the fault is a module fault. Only rack and slot values are significant.

Rack

The rack number ranges from 0 to 7. Zero is the main rack, that is, the one containing the PLC. Racks 1 through 7 are expansion racks, connected to the PLC through a Bus Transmitter Module in the main rack and Bus Receiver Modules in the expansion racks.

Slot

The slot number ranges from 0 to 9. The PLC CPU always occupies slot 1 in the main rack (rack 0).

I/O Bus

The I/O bus number ranges from 0 to 15. When the module in the slot is a single-channel Genius Bus Controller, this number is always one. When the module is an integral analog module, this designates which expansion channel generated the fault.

Block

Block refers to the serial bus address of the Genius block which reported or has the fault.

Point

Point ranges from 1 to 1024 (decimal). It tells which point on the block has the fault when the fault is a point-type fault.

I/O Fault Group

Fault group is the highest classification of a fault. It identifies the general category of the fault. The fault description text displayed by the software is based on the fault group and the error codes.

Table B-15 lists the possible fault groups in the I/O fault table. Group numbers less than 80 (Hex) are maskable faults.

The last non-maskable fault group, Additional I/O Fault Codes, is declared for the handling of new fault conditions in the system without the PLC having to specifically know the alarm codes. All unrecognized I/O-type alarm codes belong to this group.

Table B-15. I/O Fault Groups

Group Number	Group Name	Fault Action
2	Loss of, or missing, IOC	Fatal
3	Loss of, or missing, I/O module	Diagnostic
6	Addition of, or extra, IOC	Diagnostic
7	Addition of, or extra, I/O module	Diagnostic
9	IOC or I/O bus fault	Diagnostic
A	I/O module fault	Diagnostic
F	IOC software failure	Fatal
–	Additional I/O fault codes	As specified

I/O Fault Action

The fault action specifies what action the PLC CPU should take when a fault occurs. The following table lists possible fault actions.

Table B-16. PLC Fault Actions

Fault Action	Action Taken by CPU	Code
Informational	Log fault in fault table	1
Diagnostic	Log fault in fault table Set fault references	2
Fatal	Log fault in fault table Set fault references Go to Stop mode	3

I/O Fault Category

The I/O fault category specifies a general classification of the fault. It is similar to the I/O fault group, discussed earlier. I/O fault categories are listed in the following table.

Table B-17. I/O Fault Categories

Decimal Number	Hex Code	Description
1	1	Circuit fault: Short circuit, open wire, etc.
2	2	Loss of block: Block no longer responding
3	3	Addition of block: New block appeared
4	4	Unused category
5	5	Unused category
6	6	Genius bus fault
7	7	Global memory fault
8	8	EEPROM fault, watchdog timeout
9	9	Addition of IOC
10	A	Loss of, or missing, IOC
11	B	IOC software fault
12	C	Forced circuit: A Genius I/O point was forced with the HHM
13	D	Unforced circuit: HHM force was removed
14	E	Loss of Series 90 integral card
15	F	Addition of Series 90 integral card
16	10	Found extra Series 90 integral card
17	11	Found extra Genius block
18	12	An IOC detected a hardware failure or a baud rate mismatch
19	13	Genius bus controller has stopped reporting faults because too many faults have occurred
20	14	Configuration mismatch fault for I/O modules
21	15	GBC software exception
22	16	Redundant Genius block switched buse
23	17	Block not active on redundant bus

I/O Fault Type

The I/O fault type component creates sub-categories under the circuit fault, module fault, I/O bus fault, loss of block, excessive faults, I/O configuration mismatch, and GBC software exception categories. It is undefined for other fault categories, but is always set to zero when the fault category is something other than these seven categories. Table B-18 lists the I/O fault types.

Table B-18. I/O Fault Types

Number	Description
I/O Fault Types for the Circuit Fault Category	
1	Circuit fault on a discrete I/O point
2	Circuit fault on an analog I/O channel
3	Fault on a GENA
4	Fault on a low-level analog input channel
5	Fault on Remote I/O Scanner
I/O Fault Types for the Module Fault Category	
0	Block Fault (EEPROM, watchdog)
1	Analog to digital communications fault or calibration error
5	User scaling error caused out of range values
I/O Fault Types for I/O Bus Fault Category	
0	Genius IOC disabled all outputs on the bus because communications timed out between the PLC CPU and the Genius IOC
1	Genius Bus fault (No interrupt to the GBC for its turn on the bus within the time-out period)
3	Genius IOC detected a conflict between its SBA and another device on the bus
I/O Fault Types for the Loss of Block Category	
0	No reason specified.
1	Loss of A/D communications.
I/O Fault Types for Excessive Faults Category	
1	Genius IOC detected a high error count on the Genius I/O Bus and dropped off the bus for at least 1.5 seconds
I/O Fault Types for Configuration Mismatch Category	
2	Model number mismatch detected by I/O scanner
3	Non-existent I/O module detected by I/O scanner
4	I/O type mismatch detected by I/O scanner
8	Integral analog module detected; expansion analog module mismatch
9	Broadcast Data Length mismatch
A	A configured feature is not supported

Table B-18. I/O Fault Types - Continued

	I/O Fault Types for GBC Software Exception Category
1	Incoming datagram queue is full
2	The queue for Read/Write requests in the GBC is full. The requests may be from the Genius Bus or from COMMREQs
3	The low priority mail queue from the GBC to the PLC is full. The response to the PLC was lost.
4	Genius background message requiring PLC action was received before PLC completed initialization. Message was ignored.
5	Genius Report Fault message was not processed because GBC software revision is too old.
6	Excessive usage of internal GBC memory. User should verify COMMREQ usage.

I/O Fault Description

The I/O fault description component 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 undefined for other faults, but is always set to zero. The next two tables list the possible fault descriptions.

Table B-19. I/O Fault Descriptions

Number	Description
	Fault Descriptions for Discrete Circuit Faults
01	Loss of user side power.
02	Short in user wiring (for Genius, current level greater than 20 Amps).
04	Sustained overcurrent (for Genius, current level greater than 2 Amps).
08	Very low or no current flow.
10	Switch temperature too high.
20	Genius smart switch failure.
83	Series 90-70 I/O individual point fault (also indicated for I/O Module Fault category).
84	Series 90-70 output fuse blown (also indicated for I/O Module Fault category).
	Fault Descriptions for Analog Circuit Faults
01	Input channel low alarm.
02	Input channel high alarm.
04	Input channel under range.
08	Input channel over range.
10	Open wire detected on input channel.
20	Output channel under range.
40	Output channel over range.
80	Expansion channel not responding.
80	Feedback error for Genius Current Source Analog Block.

Table B-19. I/O Fault Descriptions - Continued

Number	Description
*Fault Descriptions for Low-level Analog Circuit Faults ***	
20	Improper RTD connection or thermocouple reverse junction fault.
40	Cold junction sensor fault on thermocouple block or internal error in RTD block.
80	Input channel shorted (Genius RTD and Strain Gauge Blocks only).
*Fault Descriptions for Module Faults ****	
08	Genius EEPROM or NVRAM failure.
20	Genius calibration memory failure.
40	Genius shared RAM fault.
80	Genius internal circuit fault.
81	Watchdog timeout (discrete I/O modules only).
82	Aux fault on discrete I/O modules.
83	Series 90-70 I/O individual point fault (also indicated for CIRCUIT_FAULT category).
84	Series 90-70 output fuse blown (also indicated for CIRCUIT_FAULT category).
Fault Descriptions for GENA Faults	
80	Fault on a GENA analog or discrete point.
87	Fault on Remote I/O Scanner

*   The following analog faults also apply when the low-level analog fault type is indicated: AI_LOW_ALARM, AI_HI_ALARM, AI_UNDER_RANGE, AI_OVER_RANGE, and OPEN_WIRE.

**   These faults are reported under the HEADEND_FAULT fault type.

I/O Fault Specific Data

An I/O fault table entry may contain up to 21 bytes of I/O fault specific data. In general, this area contains additional information related to the fault. Not all entries contain I/O fault specific data. This section describes those that do. All but one of these faults uses five bytes of I/O fault specific data; the global memory fault uses the 21-byte entry. If a fault is not listed, it does not have I/O fault specific data.

Faults originated by the Genius Bus Controller always have at least one byte of I/O fault specific data. This byte is in addition to whatever other data might be present.

Circuit Fault

Circuit fault entries use one or two bytes of the fault specific data area. When the Genius Bus Controller reports the fault, the first byte is generated by the Genius Bus Controller. 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. If the fault type is not a GENA fault, the second byte contains the circuit configuration and is encoded, as shown in table B-20.

Loss/Addition of Block

In the case of a Loss of Block or Addition of Block fault, four bytes of fault specific data are used. The first byte is encoded, as shown in table B-20. The second byte contains the block configuration and is encoded as shown in table B-20. The third byte specifies the number of input circuits possibly used, and the fourth byte specifies the number of output circuits possibly used.

Global Memory Fault

The Global Memory fault uses 10 data bytes. The first byte contains the subnet group number. The other nine bytes contain the global variable name, formatted as a null-terminated string.

Forced/Unforced Circuit

Three bytes of fault specific data are present when a circuit force is added or removed (Forced circuit or Unforced circuit). The first byte contains a code from table B-20. The second byte contains the block configuration, and byte 3 contains the discrete/analog indication.

Loss of or Missing IOC

When the PLC CPU registers a Loss of or Missing IOC fault, it includes in the I/O fault specific data one of the codes in table B-20.

Other I/O Faults

In addition to those faults listed above, when the Genius Bus Controller reports one of the following faults, it includes one of the bytes in table B-20 in the I/O fault specific data.

- I/O Bus fault
- Module fault
- IOC software fault
- Extra Genius block
- IOC hardware fault
- Excessive faults
- GBC software exception

Block Switch

When the fault category is Block Switch, five bytes of fault specific data are used. The first byte is encoded, as shown in table B-20. The second byte contains the block configuration and is encoded as shown in table B-20. The third byte specifies the number of input circuits possibly used, and the fourth byte specifies the number of output circuits possibly used. The last byte (byte 5) of fault specific data contains the rack/slot address of the Genius bus controller which controlled the bus that the block switched off. It is encoded with the slot number in the low four bits and the rack number in the high four bits.

Symbolic Fault Specific Data

The following table lists data that is required for four kinds of fault specific data:
- Block circuit configuration
- Block usage indication
- Discrete/analog indication
- Loss of IOC error code

Table B-20. I/O Fault Specific Data

Decimal Number	Hex Code	Description
		Circuit Configuration
	1	Circuit is an input
	2	Circuit is an output
	3	Circuit is an output with feedback
		Block Configuration
	1	Block configured for inputs only
	2	Block configured for outputs only
	3	Block has both inputs and outputs
		Discrete/Analog Indication
	1	Discrete block
	2	Analog block
		LOSS_OF_IOC Error Code
1	1	IOC failed to respond to a CPU request
2	2	CPU and IOC lost synchronization
3	3	CPU/IOC communications failed
4	4	VME bus error
5	5	VME bus error
6	6	CPU/IOC communications failed
7	7	CPU/IOC communications failed
8	8	IOC failed to respond to a CPU request
9	9	CPU/IOC communications failed
10	A	VME bus error
11	B	BME bus error
12	C	CPU/IOC communications failed
13	D	CPU/IOC communications failed
14	E	CPU/IOC communications filed.
15	F	IOC failed to respond to a CPU request
16	10	CPU/IOC communications failed
17	11	CPU/IOC communications failed
18	12	IOC failed to respond to a CPU request
19	13	CPU/IOC communications failed
20	14	CPU/IOC communications failed
21	15	Internal I/O scanner fault detected
22	16	IOC failed to respond to a CPU request
23	17	IOC failed to respond to a CPU request
24	18	IOC failed to respond to a CPU request
25	19	IOC failed to respond to a CPU request
26	1A	IOC failed to respond to a CPU request
27	1B	CPU/IOC communications failed
28	1C	VME bus error occurred while reading input data

Table B-20. I/O Fault Specific Data - Continued

Decimal Number	Hex Code	Description
29	1D	VME bus error occurred while reading input diagnostics
30	1E	CPU/IOC communications failed
31	1F	VME bus error occurred while writing output data to IOC
32	20	CPU/IOC communications failed
33	21	CPU/IOC communications failed
34	22	CPU/IOC communications failed
35	23	VME bus error
36	24	VME bus error
37	25	Unable to read data from IOC for redundant I/O blocks
38	26	Unable to write data to IOC for redundant I/O blocks
39	27	IOC does not support configured I/O redundancy
40	28	IOC failed to respond to a CPU request
41	29	I/O scanner detected too many IOCs in the system
42	2A	I/O scanner detected too many IOCs in the system
43	2B	VME bus error
44	2C	VME bus error

Fault Actions for Specific Faults

Forced/unforced circuit faults are reported as informational faults. All others are diagnostic or fatal.

The model number mismatch, I/O type mismatch and non-existent I/O module faults are reported in the PLC fault table under the System Configuration Mismatch group. They are not reported in the I/O fault table.

I/O Fault Time Stamp

The six-byte time stamp is the value of the system clock when the fault was recorded by the PLC CPU. Values are coded in BCD format.

Table B-21. I/O Fault Time Stamp

Byte Number	Description
1	Seconds.
2	Minutes.
3	Hours.
4	Day of the month.
5	Month.
6	Year.

Loss of I/O Block Example

The loss of I/O Block fault is described below. (All data is in hexadecimal.)

Field	Value	Description
Long/Short	02	This fault contains up to 5 bytes of fault specific data.
Reference Address	FF0100	FF indicates that the reference address is not given. The fault applies to the entire block.
Rack	00	Main rack (rack 0).
Slot	04	Slot 4. In this configuration, slot 4 contains a Genius Bus Controller.
Bus	01	Bus 1 on the bus controller. When this byte is significant, the single channel bus controller always shows this field as a 1.
Block	1E	1E is 30 decimal. This fault was logged for the Genius device at serial bus address 30.
Point	FF7F	FF in the first (low byte) indicates that the point is not meaningful for this fault entry.
Fault Group	03	Loss of, or missing, I/O module fault.
Fault Action	02	Diagnostic fault.
Fault Category	02	Loss of block.
Fault Type	00	00 fault type in the Loss of Block category indicates no specific fault was given.
Fault Description	00	No fault description information is given.
Fault Specific Data	84	GBC reported a lost device. Only 1 byte of fault specific data is significant.

The Genius Bus Controller in slot 4 determined that the device at serial bus address 30 on bus 1 failed to send data in three consecutive bus scans. The bus controller then marked the device as lost and logged a fault in the I/O fault table. The fault type and fault description are not meaningful on this fault. The fault specific data contains a byte from the GBC echoing the fault.

```
02  FF0100  0004011EFF7F  0302  02  00  00  8403101000
                                                        ┌─── Fault Specific Data
                                                        ├─── Fault Description
                                                        ├─── Fault Type
                                                        ├─── Fault Category
                                                        ├─── Fault Action
                                                        ├─── Fault Group
                                                        ├─── Point
                                                        ├─── Block
                                                        ├─── I/O Bus
                                                        ├─── Slot
                                                        ├─── Rack
                                                        ├─── Reference Address
                                                        └─── Long/Short
```
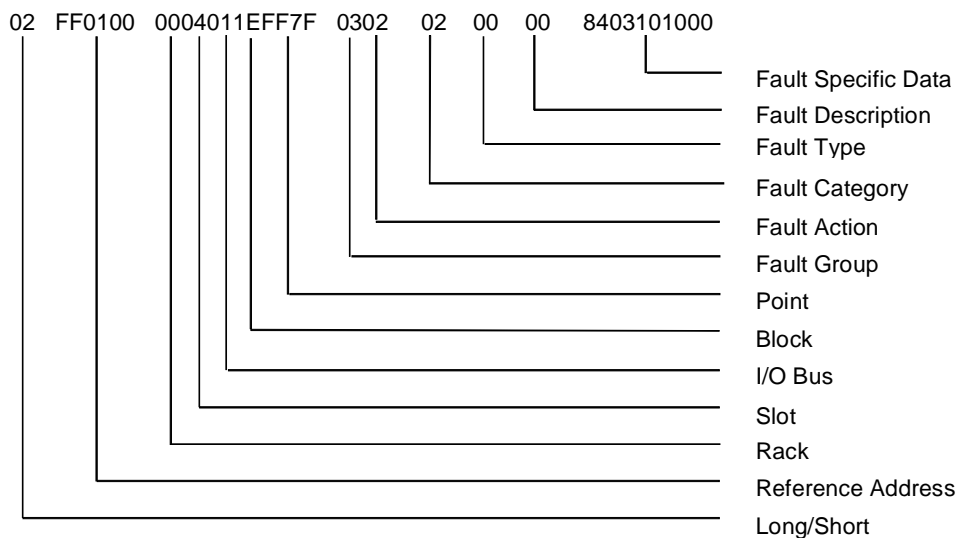
Addition of I/O Block Example

The Addition of I/O Block fault is described below. (All data is in hexadecimal.)

Field	Value	Description
Long/Short	02	This fault contains up to 5 bytes of fault specific data.
Reference Address	1F0100	1F (31 decimal) indicates that the block has a discrete grouped reference address. The software shows this by displaying a %QI in the reference address column. 0100 indicates this is address 0001 (hexadecimal data is displayed low byte first, then high byte) in the discrete grouped address space.
Rack	00	Main rack (rack 0).
Slot	04	Slot 4. In this configuration, slot 4 contains a Genius Bus Controller.
Bus	01	Bus 1 on the bus controller. When this byte is significant, the single channel bus controller always shows this field as a 1.
Block	1E	1E is 30 decimal. This fault was logged for the Genius device at serial bus address 30.
Point	FF7F	FF in the first (low byte) indicates that the point is not meaningful for this fault entry.
Fault Group	07	Addition of, or extra, I/O module fault.
Fault Action	02	Diagnostic fault.
Fault Category	03	Fault category 03 is an addition of block.
Fault Type	00	No fault type data occurs for an Addition of Block fault. The 00 is meaningless.
Fault Description	00	No fault description information is given.
Fault Specific Data	80	GBC reported an added device. Only 1 byte of fault specific data is significant.

The bus controller in slot 4 in the main rack received data from the device at serial bus address 30, indicating that the device was again on the bus. From the configuration table stored from your programming software to the PLC CPU, the CPU determined that the first point on the device was %QI0001. The fault type and fault description are not meaningful on this fault. The fault specific data contains a byte from the GBC echoing the fault.

```
02  1F0100  0004011EFF7F  0702  03  00  00  8003101000
                                                    │──── Fault Specific Data
                                                    │──── Fault Description
                                                    │──── Fault Type
                                                    │──── Fault Category
                                                    │──── Fault Action
                                                    │──── Fault Group
                                                    │──── Point
                                                    │──── Block
                                                    │──── I/O Bus
                                                    │──── Slot
                                                    │──── Rack
                                                    │──── Reference Address
                                                    │──── Long/Short
```
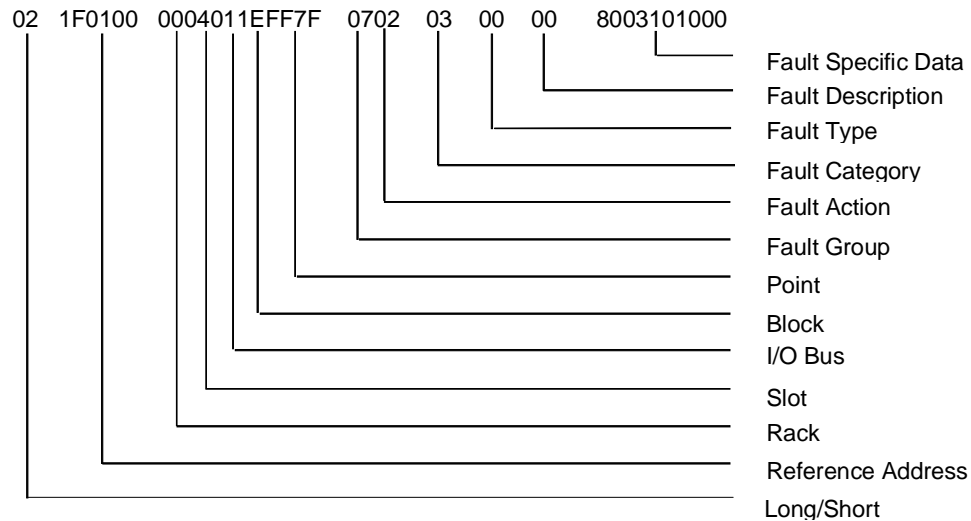
Loss of IOC (I/O Controller)

The Loss of IOC fault is explained below. (All values are in hexadecimal.)

Field	Value	Description
Long/Short	02	This fault contains up to 5 bytes of fault specific data.
Reference Address	000000	All zeros in this field indicates that the reference address is not meaningful for this fault.
Rack	00	Main rack (rack 0).
Slot	03	Slot 3. In this configuration, slot 3 contains a Genius Bus Controller.
Bus	7F	Since this is a single channel GBC, the bus number is not needed. 7F indicates that the bus number is not significant.
Block	7F	A block number of 7F hex indicates that the block is not meaningful for this fault.
Point	FF7F	FF in the first (low byte) indicates that the point is not meaningful for this fault entry.
Fault Group	02	Loss of, or missing, IOC fault.
Fault Action	03	Fatal fault.
Fault Category	0A	Fault Category 0A (10 decimal) is a Loss of, or Missing, IOC fault.
Fault Type	00	No fault type data occurs for a Loss of, or Missing, IOC fault.
Fault Description	00	No fault description information is given.
Fault Specific Data	00	There is no entry for a zero in fault specific data, so there is no additional information available on this instance of LOSS of IOC.

The PLC CPU detected a loss of, or missing, IOC and logged this fault. The fatal action indicates that the PLC CPU will not transition to RUN mode until the fault is cleared. Fault type and fault description are not meaningful for this fault; fault specific data may be meaningful.

```
02   000000   00037F7FFFF7F   0203   0A   00   00   0002400000
                                                              └─── Fault Specific Data
                                                         └──────── Fault Description
                                                    └───────────── Fault Type
                                             └──────────────────── Fault Category
                                        └───────────────────────── Fault Action
                                 └────────────────────────────────── Fault Group
                          └───────────────────────────────────────── Point
                        └────────────────────────────────────────────── Block
                     └──────────────────────────────────────────────────── I/O Bus
                   └────────────────────────────────────────────────────────── Slot
                 └──────────────────────────────────────────────────────────────── Rack
          └───────────────────────────────────────────────────────────────────────── Reference Address
     └────────────────────────────────────────────────────────────────────────────────── Long/Short
```
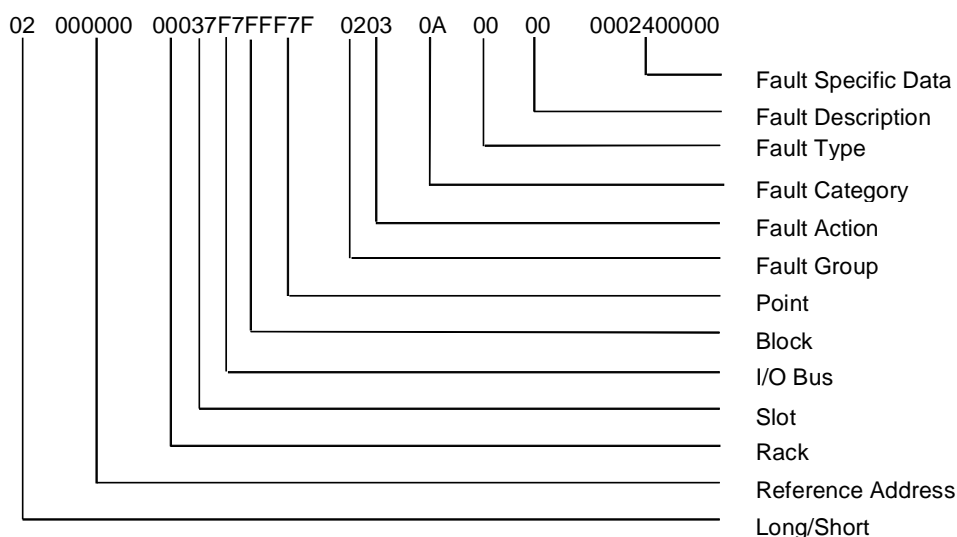
Circuit Fault

The Circuit Fault is explained below. (All values are in hexadecimal.)

Field	Value	Description
Long/Short	02	This fault contains up to 5 bytes of fault specific data.
Reference Address	0D0200	0D (13 decimal) indicates that the block has an analog grouped reference address. Your programming software shows this by displaying a %AQI in the reference address column. 0200 indicates this is address 0002 (hexadecimal data is displayed low byte first, then high byte) in the analog grouped address space.
Rack	00	Main rack (rack 0).
Slot	03	Slot 3. In this configuration, slot 3 contains a Genius Bus Controller.
Bus	01	Bus 1 on the bus controller. When this byte is significant, the single channel bus controller always shows this field as a 1.
Block	02	This fault was logged for the Genius device at serial bus address 02.
Point	0200	This fault was logged for point 2 on the device at serial bus address 02.
Fault Group	0A	0A hexadecimal is 10 decimal. I/O Module fault.
Fault Action	02	Diagnostic fault.
Fault Category	01	Fault Category 01 is a circuit fault.
Fault Type	02	Fault Type 02 under the Circuit Fault category is a circuit fault on an analog I/O channel.
Fault Description	10	A fault description of 10 hex under the analog circuit fault description is open wire.
Fault Specific Data	9B03	The GBC reported a circuit fault. Only the 9B byte is significant.

The Genius Bus Controller in slot 3 in the main rack received a circuit fault "Report Fault" message from the analog block located at serial bus address 2. The block reported an Open Wire fault on point 2. From the configuration tables stored from your programming software, the PLC CPU determined that the analog block at serial bus address 02 was mapped to both the %AI and %AQ user references and that point 2 corresponded to location 0002.

```
02  0D0200  000301020200  0A02  01  02  10  9B03000000
                                                      │──── Fault Specific Data
                                                  │──────── Fault Description
                                              │──────────── Fault Type
                                          │──────────────── Fault Category
                                      │──────────────────── Fault Action
                                  │──────────────────────── Fault Group
                                                            Point
                                                            Block
                                                            I/O Bus
                                                            Slot
                                                            Rack
                                                            Reference Address
                                                            Long/Short
```
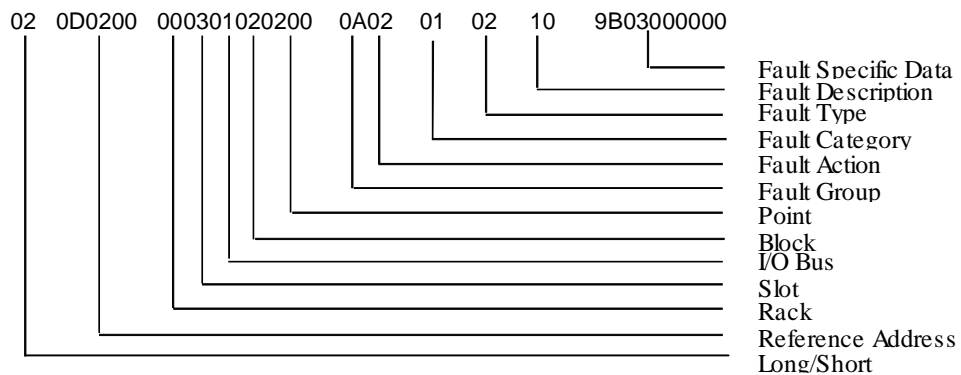
Instruction Mnemonics

In program display/edit mode, you can quickly enter or search for a program instruction by typing the ampersand (&) character followed by the instruction's mnemonic. For some instructions, you can also specify a reference address or nickname, a label, or a location reference address.

This appendix lists the mnemonics of the programming instructions for Logicmaster 90-70 programming software. At any time during programming, you can display a help screen with these mnemonics by pressing the **ALT** and **I** keys.

Table C-1. Contacts, Coils, Links, Timers, and Counters

Instruction	Mnemonic									
	All	**INT**	**UINT**	**DINT**	**BIT**	**BYTE**	**WORD**	**DWORD**	**REAL**	**MIXED**
Contacts										
Any Contact	&CON	&CON								
–| |–	&NOCON	&NOCON								
–|/|–	&NCCON	&NCCON								
–|↑|–	&PCON	&PCON								
–|↓|–	&NCON	&NCON								
–[FAULT]–	&FA	&FA								
–|NOFLT|–	&NOF	&NOF								
–[HIALR]–	&HI	&HI								
–[LOALR]–	&LOA	&LOA								
<+>——	&CONC	&CONC								
Coils										
Any Coil	&COI	&COI								
–( )–	&NOCOI	&NOCOI								
–(/)–	&NCCOI	&NCCOI								
–(↑)–	&PCOI	&PCOI								
–(↓)–	&NCOI	&NCOI								
–(S)–	&SL	&SL								
–(r)–	&RL	&RL								
–(SM)–	&SM	&SM								
–(RM)–	&RM	&RM								
–(M)–	&NOM	&NOM								
–(/M)–	&NCM	&NCM								
——<+>	&COILC	&COILC								
Links										
Horizontal	&HO	&HO								
Vertical	&VE	&VE								
Timers										
ONDTR	&ON	&ON								
OFDT	&OF	&OF								
TMR	&TM	&TM								
Counters										
UPCTR	&UP	&UP								
DNCTR	&DN	&DN								

Table C-2. Math, Relational, and Bit Operations

Instruction	Mnemonic									
	All	INT	UINT	DINT	BIT	BYTE	WORD	DWORD	REAL	MIXED
Math										
ADD	&AD	&AD_I	&AD_UI	&AD_DI					&AD_R	
SUB	&SUB	&SUB_I	&SUB_UI	&SUB_DI					&AD_R	
MUL	&MUL	&MUL_I	&MUL_UI	&MUL_DI					&MUL_R	&MUL_M
DIV	&DIV	&DIV_I	&DIV_UI	&DIV_DI					&DIV_R	&DIV_M
MOD	&MOD	&MOD_I	&MOD_UI	&MOD_DI						
SQRT	&SQ	&SQ_I		&SQ_DI					&SQ_R	
ABS	&ABS	&ABS_I		&ABS_DI					&ABS_R	
SIN	&SIN	&SIN								
COS	&COS	&COS								
TAN	&TAN	&TAN								
IASIN	&ASIN	&ASIN								
ACOS	&ACOS	&ACOS								
ATAN	&ATAN	&ATAN								
LOG	&LOG	&LOG								
LN	&LN	&LN								
EXP	&EXP	&EXP								
EXPT	&EXPT	&EXPT								
DEG	&DEG	&DEG								
RAD	&RAD	&RAD								
Relational										
EQ	&EQ	&EQ_I	&EQ_UI	&EQ_DI					&EQ_R	
NE	&NE	&NE_I	&NE_UI	&NE_DI					&NE_R	
GT	&GT	&GT_I	&GT_UI	&GT_DI					&GT_R	
GE	&GE	&GE_I	&GE_UI	&GE_DI					&GE_R	
LT	&LT	&LT_I	&LT_UI	&LT_DI					&LT_R	
LE	&LE	&LE_I	&LE_UI	&LE_DI					&LE_R	
CMP	&CM	&CM_I	&CM_UI	&CM_DI					&CM_R	
Bit Operation										
AND	&AN						&AN_W	&AN_DW		
OR	&OR						&OR_W	&OR_DW		
XOR	&XO						&XO_W	&XO_DW		
NOT	&NOT						&NOT_W	&NOT_DW		
SHL	&SHL						&SHL_W	&SHL_DW		
SHR	&SHR						&SHR_W	&SHR_DW		
ROL	&ROL						&ROL_W	&ROL_DW		
ROR	&ROR						&ROR_W	&ROR_DW		
BTST	&BT						&BT_W	&BT_DW		
BSET	&BS						&BS_W	&BS_DW		
BLCR	&BCL						&BCL_W	&BCL_DW		
BPOS	&BP						&BP_W	&BP_DW		
MCMP	&MCM						&MCM_W	&MCM_DW		

Table C-3. Data Move and Data Table Operations

Instruction	Mnemonic									
	All	**INT**	**UINT**	**DINT**	**BIT**	**BYTE**	**WORD**	**DWORD**	**REAL**	**MIXED**
Data Move										
MOVE	&MOV	&MOV_I	&MOV_UI	&MOV_DI	&MOV_BI		&MOV_W	&MOV_DW	&MOV_R	
BLKMOV	&BLKM	&BLKM_I	&BLKM_UI	&BLKM_DI			&BLKM_W	&BLKM_DW	&BLKM_R	
BLKCLR	&BLKC									
SHFR	&SHF				&SHF_BI		&SHF_W	&SHF_DW		
BITSEQ	&BI									
SWAP	&SW						&SW_W	&SW_DW		
COMMREQ	&COMMR									
VMERD	&VMERD					&VMERD_BY	&VMERD_W			
VMEWRT	&VMEW					&VMEWR_BY	&VMERD_W			
VMERMW	&VMERM					&VMERM_BY	&VMERM_W			
VMETST	&VMET					&VMET_BY	&VMET_W			
VME_CFG_RD	&CFGRD									
VME_CFG_WRT	&CFGWRT									
DATA_INIT	&DINI									
DATA_INIT_COMM	&DCO									
DATA_INIT_ASCII	&DA									
Data Table										
TBLRD	&TBLR	&TBLR_I	&TBLR_UI	&TBLR_DI			&TBLR_W	&TBLR_DW		
TBLWR	&TBLW	&TBLW_I	&TBLW_UI	&TBLW_DI			&TBLW_W	&TBLW_DW		
LIFORD	&LIFOR	&LIFOR_I	&LIFOR_UI	&LIFOR_DI			&LIFOR_W	&LIFOR_DW		
LIFOWRT	&LIFOW	&LIFOW_I	&LIFOW_UI	&LIFOW_DI			&LIFOW_W	&LIFOW_DW		
FIFORD	&FIFOR	&FIFOR_I	&FIFOR_UI	&FIFOR_DI			&FIFOR_W	&FIFOR_DW		
FIFOWRT	&FIFOW	&FIFOW_I	&FIFOW_UI	&FIFOW_DI			&FIFOW_W	&FIFOW_DW		
SORT	&SORT	&SORT_I	&SORT_UI				&SORT_W			
ARRAY_MOVE	&AR	&AR_I	&AR_UI	&AR_DI	&AR_BI	&AR_BY	&AR_W	&AR_DW		
SRCH_EQ	&SRCHE	&SRCHE_I	&SRCHE_UI	&SRCHE_DI		&SRCHE_BY	&SRCH_W	&SRCH_DW		
SRCH_NE	&SRCHN	&SRCHN_I	&SRCHN_UI	&SRCHN_DI		&SRCHN_BY	&SRCHN_W	&SRCHN_DW		
SRCH_GT	&SRCHGT	&SRCHGT_I	&SRCHGT_UI	&SRCHGT_DI		&SRCHGT_BY	&SRCHGT_W	&SRCHGT_DW		
SRCH_GE	&SRCHGE	&SRCHGE_I	&SRCHGE_UI	&SRCHGE_DI		&SRCHGE_BY	&SRCHGE_W	&SRCHGE_DW		
SRCH_LT	&SRCHLT	&SRCHLT_I	&SRCHLT_UI	&SRCHLT_DI		&SRCHLT_BY	&SRCHLT_W	&SRCHLT_DW		
SRCH_LE	&SRCHLE	&SRCHLE_I	&SRCHLE_UI	&SRCHLE_DI		&SRCHLE_BY	&SRCHLE_W	&SRCHLE_DW		

Table C-4. Conversion and Control Operations

Instruction	Mnemonic									
	All	INT	UINT	DINT	BIT	BYTE	WORD	DWORD	REAL	MIXED
Conversion										
to BCD–4	&BCD4	&BCD4	&BCD4_UI							
to BCD–8				&BCD8						
to UINT	&UI	&UI		&UI_DI					&UI_R	
to INT	&I	&I		&I_DI					&I_R	
to DINT	&DIN	&DIN	&DIN_UI						&DIN_R	
to REAL	&R	&R	&R_UI	&R_DI						
BCD–4 to UINT	&UI_BCD4									
BCD–4 to INT	&I_BCD4									
BCD–8 to DINT	&DI_BCD8									
BCD–4 to REAL	&R_BCD4									
BCD–8 to REAL	&R_BCD8									
TRUN	&TRINT	&TRINT		&TRDINT						
Control										
CALL	&CA									
DOIO	&DO									
SUSIO	&SUS									
MCR	&MCR									
ENDMCR	&ENDMCR									
JUMP	&JUMP									
LABEL	&LABEL									
COMMENT	&COMME									
SVCREQ	&SV									
PIDISA	&PIDIS									
PIDIND	&PIDIN									
FOR	&FOR									
END_FOR	&ENDFOR									
EXIT	&EXIT									

Appendix D

Memory Allocation

CPU Memory Size is the number of bytes of memory available to the user for PLC applications. The CPU memory size varies based on CPU and daughterboard selected. The current options are shown below.

CPU Memory Size	Bytes
32K	32,768
64K	65,536
128K	131,072
256K	262,144
512K	524,288
1M	1,048,576
6M	6,291,456

The following items count against the CPU memory size:

Register Memory Size (%R)
Bytes = %R references configured $\times$ 2

Analog Inputs (%AI)
If point faults enabled: Bytes = %AI references configured $\times$ 3

If point faults disabled: Bytes = %AI references configured $\times$ 2

Analog Outputs (%AQ)
If point faults enabled: Bytes = %AQ references configured $\times$ 3

If point faults disabled: Bytes = %AQ references configured $\times$ 2

Discrete Point Faults
If point faults enabled:

Bytes = 128 for 73x CPUs

Bytes = 512 for 77x CPUs

Bytes = 3072 for 78x and 9xx CPUs

Fault Tables

Bytes = (I/O Fault Entries configured + PLC Fault entries configured – 48) × 48

Note

The default fault table sizes (16 PLC faults, 32 I/O faults) do not require user memory

Ethernet Global Data

Bytes = 0 if no Ethernet Global Data exchanges are configured

C Debugger Connection

Bytes = 11772 when connection is made; no user memory used when no connection

I/O Scan Set File

Based on number of scan sets used.

Note

32 bytes of user memory is consumed if the user scans all I/O every sweep (the default).

Module Configuration Files

Applies to FBC module only

Name Resolution Files

Bytes = 0 if no name resolution file created by user

User Protocol Files

Bytes = 0 if no user protocol files are created by user

User Programs

See description below on user programs

There are some additional charges for Redundancy products such as the 788, 789, and CGR935 CPUs. Refer to the appropriate user manuals for details.

User Program Memory Usage

The user memory remaining after subtracting the configuration items above from the CPU memory size can all be devoted to user logic. However, multiple programs may be needed to take advantage of this memory for CPUs with a 1M or higher CPU memory size.

There is a separate maximum program size for each user program created. This size varies based on program type. The maximum sizes are shown in the table below. Note that the maximum size is only possible on CPUs with 1M or more of user memory; for CPUs with less memory, the maximum size is limited to the CPU memory size.

Type of Program	Stack Size	Maximum Program Size *
LD/SFC Program	≤ 12K (default size is 12K)	544K Stack size is not counted against program size or total user memory.
LD/SFC Program	> 12K	544K – Stack Size
Standalone C Program	N/A	564K – Stack Size

*Maximum program size is only possible on CPUs with 1M byte or more of user memory; otherwise, maximum program size is limited to size of CPU memory.

The following User Program items are counted against the CPU memory size but *not* against the User Program size for the associated program:

C Initialization Data

The initialization data for the data areas of C Standalone programs and EXE Blocks is kept separate from the rest of the program image and is not counted against the user program limit. This memory does count against the CPU memory size. Refer to the *C Programmer's Toolkit for Series 90-70 PLCs User's Manual* for details.

Standalone C Program Control Structures

8K (8192 bytes) is charged per Standalone C program created and stored to the CPU

The following User Program items are counted against the User Program limit for the associated program and also against the CPU memory size.

%P Program Memory [1]

If highest %P used is %P00128 or less:
Bytes = 399

If highest %P used is greater than %P00128:
Bytes = Highest %P used (rounded to next multiple of 32) $\times 2 + 143$

---

[1] To round to next multiple of 32 for %P and %L: Divide # of references by 32 and round up to a whole number, then multiply by 32. For example, 167 %P references yields 192 since 167 / 32 = 5.21875 which rounds up to 6 and 6 X 32 = 192.

%L Program Block Memory Usage[1]

For each program block,

If highest %L used is %L0064 or less:
Bytes = 271
If highest %L used is greater than %L0064:
Bytes = Highest %L used (rounded to next multiple of 32) $\times$ 2 + 143

Interrupt Block Control Structures

142 bytes per interrupt block

Program Logic and Overhead

C Data Area

The data area for standalone C programs and EXE blocks is considered part of the user program and counts against the user program size

Configured Stack Size

Shown in program limits table above.

Note

The first 12K of the LD/SFC program's stack is not counted against the CPU's memory size

Also, note that the program block is based on overhead for the block itself plus the logic and register data being used (that is, %L). Refer to the Call instruction in Table A-1 (Appendix A) for block size overhead.

Note

If the user has a 1M or larger CPU and needs more space for LD/SFC logic, then they may want to consider changing some of their %P or %L references to %R, %AI, or %AQ. %R, %AI, and %AQ references only count against the CPU memory size and not against the program limit. Such changes require a recompilation of the program block and a stop mode store to the CPU.

---

[1] To round to next multiple of 32 for %P and %L: Divide # of references by 32 and round up to a whole number, then multiply by 32. For example, 167 %P references yields 192 since 167 / 32 = 5.21875 which rounds up to 6 and 6 X 32 = 192.

The following table lists the keyboard functions that are active in the Logicmaster 90-70 software environment. This information may also be displayed on the programmer screen by pressing ALT-K to access key help.

Key Sequence	Description	Key Sequence	Description
Keys Available throughout the Software Package			
ALT-A	Abort.	CTRL-Break	Exit package.
ALT-C	Clear field.	Esc	Zoom out.
ALT-M	Change programmer mode.	CTRL-Home	Previous command line contents.
ALT-R	Change PLC **RUN/STOP** state.	CTRL-End	Next command line contents.
ALT-E	Toggle status area.	CTRL-←	Cursor left within the field.
ALT-L	List directory files.	CTRL-→	Cursor right within the field.
ALT-P	Print screen.	CTRL-D	Decrement reference address.
ALT-H	Help.	CTRL-U	Increment reference address.
ALT-K	Key help.	Tab	Change/increment field contents.
ALT-I	Instruction mnemonic help.	Shift-Tab	Change/decrement field contents.
ALT-T	Start **TEACH** mode.	Enter	Accept field contents.
ALT-Q	Stop **TEACH** mode.	CTRL-E	Display last system error.
ALT-Z	Pause **TEACH** playback.	F12 (or Keypad–)	Toggle discrete reference.
ALT-n	Playback file n (n = 0 thru 9).	F11 (or Keypad *)	Override discrete reference.
Keys Available in the Program Editor Only			
ALT-B	Toggle text editor bell.	Keypad +	Accept rung.
ALT-D	Delete rung element./Delete rung.	Enter	Accept rung.
ALT-S	Store block to PLC and disk.	CTRL-PgUp	Previous rung.
ALT-X	Display zoom level & block state.	CTRL-PgDn	Next rung.
ALT-V	Variable table window.	~	Horizontal link.
ALT-W	Display PSB parameter table.	|	Vertical link.
ALT-F2	Go to operand reference table.	Tab	Go to next operand field.
Special Keys			
ALT-G	Single sweep debug. Available only in program editor, reference table editor, and status fault tables.		
ALT-N	Toggle display options. Available only in program editor and reference table editor.		
ALT-O	Password override. Available only on Password screen in configuration software.		
ALT-U	Update disk.		

The Help card on the next page contains a listing of the key help and also the instruction mnemonics help text for Logicmaster 90 software. This card is printed in triplicate and is perforated for easier removal from the manual.

This page contains side 1 of GFJ-056B.

This page contains side 2 of GFJ-056B.

Using Floating-Point Numbers

FloatingPoint Numbers

Your programming software provides the ability to edit, display, store, and retrieve numbers with real values. Some functions operate on floating-point numbers. However, in order to use floating-point numbers, you must have a 732, 772, 782 , 914 or higher CPU. Floating-point numbers are represented in decimal scientific notation, with a display of six significant digits.

Note

Use of floating-point numbers within a parameterized subroutine block (PSB) usually necessitates the use of NWORD parameters rather than WORD parameters.

Note

In this manual, the terms "floating-point" and "REAL" are used interchangeably to describe the floating-point number display/entry feature of the software.

In the software, the following format is used. For numbers in the range 9999999999 to .0001, the display has no exponent and up to six or seven significant digits. For example:

Entered	Displayed	Description
.000123456789	+.0001234567	Ten digits, six or seven significant.
–12.345e-2	–.1234500	Seven digits, six or seven significant.
1234	+1234.000	Seven digits, six or seven significant.

Outside the range listed above, only six significant digits are displayed and the display has the form:

```
+1.23456E+12
|||   |  | |
|||   |  | +———— Exponent (signed power of 10)
|||   |  |
|||   |  +————————— Exponent indicator and sign of exponent
|||   |
|||   +——————————————— Five less significant digits
|||
||+————————————————— Decimal point
||
|+——————————————————— Most significant digit
|
+————————————————————— Sign of the entire number
```

Internal Format of Floating-Point Numbers

Floating-point numbers are stored in single precision IEEE-standard format. This format requires 32 bits, which translates to two (adjacent) 16-bit PLC registers. The encoding of the bits is diagrammed below.

```
|<————————— Bits 17*32 —————————>|<————————— Bits 1-16 —————————>|
|32|   |   |   |   |   |   |   |17|16|   |   |   |   |   |   |   |1|

                          |<———————————————>23-bit mantissa<——————————|

        |<—————————>|  8-bit exponent

        |<———————————————  1-bit sign (Bit 32)
```

Register use by a single floating-point number is diagrammed below. In this diagram, if the floating-point number occupies registers R5 and R6, for example, then R5 is the least significant register and R6 is the most significant register.

```
|<——————— Least Significant Register ———————>|
|<——————————— Bits 1-16 ———————————>|
|16|   |   |   |   |   |   |   |   |1| <——— Least Significant Bit:  Bit 1

        |<———————————————————————————— Most Significant Bit:  Bit 16
```

..


```
              ┌──── Most Significant Register ────►
              ├──────── Bits 17-32 ──────────────►
    ┌32┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬17┐◄─────── Least Significant Bit:  Bit 17
    │  │ │ │ │ │ │ │ │ │ │ │ │ │ │  │
    └──┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴──┘
       ◄─────────────────────────────────── Most Significant Bit:  Bit 32
```

Values of Floating-Point Numbers

Use the following table to calculate the value of a floating-point number from the binary number stored in two registers.

Exponent (e)	Mantissa (f)	Value of Floating Point Number
255	Non-zero	Not a valid number (NaN).
255	0	$-1^s * \infty$
$0 < e < 255$	Any value	$-1^s * 2^{e-127} * 1.f$
0	Non-zero	$-1^s * 2^{-126} * 0.f$
0	0	0

f = the mantissa. The mantissa is a binary fraction.

e = the exponent. The exponent is an integer E such that E+127 is the power of 2 by which the mantissa must be multiplied to yield the floating-point value.

s = the sign bit.

∗ = the multiplication operator.

For example, consider the floating-point number 12.5. The IEEE floating-point binary representation of the number is:

$$01000001\ 01001000\ 00000000\ 00000000$$

or 41480000 hex in hexadecimal form. The most significant bit (the sign bit) is zero (s=0). The next eight most significant bits are 10000010, or 130 decimal (e=130).

The mantissa is stored as a decimal binary number with the decimal point preceding the most significant of the 23 bits. Thus, the most significant bit in the mantissa is a multiple of 2^{-1}, the next most significant bit is a multiple of 2^{-2}, and so on to the least significant bit, which is a multiple of 2^{-23}. The final 23 bits (the mantissa) are:

$$1001000\ 00000000\ 00000000$$

The value of the mantissa, then, is .5625 (that is, $2^{-1} + 2^{-4}$).

Since e > 0 and e < 255, we use the third formula in the table above:

$$
\begin{aligned}
\text{number} &= -1^s * 2^{e-127} * 1.f \\
&= -1^0 * 2^{130-127} * 1.5625 \\
&= 1 * 2^3 * 1.5625 \\
&= 8 * 1.5625 \\
&= 12.5
\end{aligned}
$$

Thus, you can see that the above binary representation is correct.

The range of numbers that can be stored in this format is from ± 1.401298E–45 to ± 3.402823E+38 and the number zero.

Entering and Displaying Floating-Point Numbers

In the mantissa, up to six or seven significant digits of precision may be entered and stored; however, the software will display only the first six of these digits. The mantissa may be preceded by a positive or negative sign. If no sign is entered, the floating-point number is assumed to be positive.

If an exponent is entered, it must be preceded by the letter "E" or "e" and the mantissa must contain a decimal point to avoid mistaking it for a hexadecimal number. The exponent may be preceded by a sign, but if none is provided, it is assumed to be positive. If no exponent is entered, it is assumed to be zero. No spaces are allowed in a floating-point number.

To provide ease-of-use, several formats are accepted in both command-line and field data entry. These formats include an integer, a decimal number, or a decimal number followed by an exponent. These numbers are converted to a standard form for display once the user has entered the data and pressed the **Enter** key.

Examples of valid floating-point number entries and their normalized display are shown below.

Entered	Displayed
250	+250,0000
+4	+4.000000
–2383019	–2383019.
34.	+34.00000
–.0036209	–.003620900
12.E+9	+1.20000E+10
–.0004E–11	–4.00000E–15
731.0388	+731.0388
99.20003e–29	+9.92000E–28

Examples of invalid floating-point number entries are shown below.

Invalid Entry	Explanation
–433E23	Missing decimal point.
10e-19	Missing decimal point.
1 0.e19	The mantissa cannot contain spaces between digits or characters. This is accepted as 10.e0, and an error message is displayed.
4.1e1 9	The exponent cannot contain spaces between digits or characters. This is accepted as 4.1e0, and an error message is displayed.

Errors in Floating-Point Numbers and Operations

Overflow occurs when a number greater than 3.402823E+38 or less than –3.402823E+38 is generated by a REAL function. 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 3.402823E+38) or negative infinity (for a number less than –3.402823E+38). You can determine where this occurs by testing the sense of the Enable Out output.

POS_INF   = 7F800000h   – IEEE positive infinity representation in hex
NEG_INF   = FF800000h   – IEEE negative infinity representation in hex

If the infinities produced by overflow are used as operands to other REAL 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 for its result.

Each REAL function capable of producing an NaN produces a specialized NaN which identifies the function.

NaN_ADD   = 7F81FFFFh   – Real addition error value in hex
NaN_SUB   = 7F81FFFFh   – Real subtraction error value in hex
NaN_MUL   = 7F82FFFFh   – Real multiplication error value in hex
NaN_DIV   = 7F83FFFFh   – Real division error value in hex
NaN_SQRT   = 7F84FFFFh   – Real square root error value in hex
NaN_LOG   = 7F85FFFFh   – Real logarithm error value in hex
NaN_POW0   = 7F86FFFFh   – Real exponent error value in hex
NaN_SIN   = 7F87FFFFh   – Real sine error value in hex
NaN_COS   = 7F88FFFFh   – Real cosine error value in hex
NaN_TAN   = 7F89FFFFh   – Real tangent error value in hex
NaN_ASIN   = 7F8AFFFFh   – Real inverse sine error value in hex
NaN_ACOS   = 7F8BFFFFh   – Real inverse cosine error value in hex
NaN_BCD   = 7F8CFFFFh   – BCD-4 to real error
REAL_INDEF   = FFC00000   – Real indefinite, divide 0 by 0 error

When an NaN result is fed into another function, it passes through to the result. For example, if an NaN_ADD is the first operand to the SUB_REAL function, the result of the SUB_REAL is NaN_ADD. If both operands to a function are NaNs, the first operand will pass through. Because of this feature of propagating NaNs through functions, you can identify the function where the NaN originated.

Note

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

Index

N

O

P

R