Series 90-30/20/Micro PLC CPU Instruction Set Reference Manual

Series 90-30/20/Micro PLC CPU Instruction Set Reference Manual

GE Fanuc Automation

Programmable Control Products

Series 90™-30/20/Micro PLC CPU Instruction Set

Reference Manual

GFK-0467M *May 2002*

Warnings, Cautions, and Notes as Used in this Publication

Warning

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

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

Caution

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

Note

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

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all hardware and software systems. GE Fanuc 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.

This manual describes the system operation, fault handling, and Logicmaster 90™ programming instructions for the Series 90™-30, Series 90-20 and Series 90 Micro programmable logic controllers. Series 90-30 PLCs, Series 90-20 PLCs, and Series 90 Micro PLCs are members of the Series 90 family of programmable logic controllers from GE Fanuc Automation.

Revisions to This Manual

- Added the model 374 CPU, which supports connection to an Ethernet network through two built-in 10BaseT/100BaseTx auto-negotiating full-duplex Ethernet ports. Models 364 (release 9.10 and later) and 374 are the only Series 90-30 CPUs that support Ethernet Global Data. Note that the CPU374 is supported only by the Windows®-based programmers.
- Other corrections and clarifications as necessary.

Related Publications

Logicmaster™ 90 Series 90™-30/20/Micro Programming Software User's Manual (GFK-0466).

VersaPro™ Programming Software User's Guide (GFK-1670)

CIMPLICITY® Machine Edition Getting Started (GFK-1868)

Series 90™-30 Programmable Controller Installation Manual (GFK-0356)

Series 90™-20 Programmable Controller Installation Manual (GFK-0551)

Series 90™-30 I/O Module Specifications Manual (GFK-0898)

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

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

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

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

Series 90™-30 and 90-20 PLC Hand-Held Programmer User's Manual (GFK-0402)

Power Mate APM for Series 90™-30 PLC—Standard Mode User's Manual (GFK-0840)

Power Mate APM for Series 90™-30 PLC—Follower Mode User's Manual (GFK-0781)

Motion Mate™ DSM302 for Series 90™-30 PLCs User's Manual (GFK-1464)

Series 90™-30 High Speed Counter User's Manual (GFK-0293)

Series 90™-30 Genius Communications Module User's Manual (GFK-0412)

Series 90™-30 Genius™ Bus Controller User's Manual (GFK-1034)

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

Series 90™-30 FIP Remote I/O Scanner User's Manual (GFK-1037)

Field Control™ Distributed I/O and Control System Genius™ Bus Interface Unit User's Manual (GFK-0825)

Series 90™ Micro Programmable Logic Controller User's Manual (GFK-1065)

*Series 90™ PLC Serial Communications User's Manua*l (GFK-0582)

Chapter 1 Introduction

The Series 90-30, 90-20, and Micro PLCs are members of the GE Fanuc Series 90 family of Programmable Logic Controllers (PLCs). They are easy to install and configure, offer advanced programming features, and are compatible with the Series 90-70 PLCs.

The 341 and lower Series 90-30 PLCs and Series 90-20 PLC use an 80188 microprocessor. The 35x and 36x series of 90-30 PLCs use an 80386EX microprocessor. The 37x series of 90-30 PLCs use a 586 microprocessor. The Series 90 Micro PLC uses the H8 microprocessor. Both program execution and basic housekeeping tasks such as diagnostic routines, input/output scanners, and alarm processing are supported. The system firmware also contains routines to communicate with the programmer. These routines provide for the upload and download of application programs, return of status information, and control of the PLC.

In the Series 90-30 PLC, the application (user logic) program that controls the end process to which the PLC is applied is controlled by a dedicated Instruction Sequencer Coprocessor (ISCP). The ISCP is implemented in hardware in the Model 313 and higher and in software in the Model 311 systems, and the Micro PLC. The microprocessor and the hardware-based ISCP can execute simultaneously, allowing the microprocessor to service communications while the ISCP is executing the bulk of the application program; however, the microprocessor must execute the non-Boolean function blocks.

Faults occur in the Series 90-30 PLC, Series 90-20 PLC, and the Micro PLC when certain failures or conditions happen that affect the operation and performance of the system. These conditions may affect the ability of the PLC to control a machine or process. Other conditions may only act as an alert, such as a low battery signal to indicate that the voltage of the battery protecting the memory is low and should be replaced. The condition or failure is called a fault.

Faults are handled by a software alarm processor function that records the faults in either the PLC fault table or the I/O fault table. (Model 331 and higher CPUs also time-stamp the faults.) These tables can be displayed through the programming software on the PLC Fault Table and I/O Fault Table screens in Logicmaster 90-30/20/Micro software using the control and status functions.

Note

Floating-point capabilities are ***only*** supported on the 35x and 36x series CPUs Release 9 or later, and on all releases of CPU352 and CPU374.

The CPU364 (release 9.10 or later) and the CPU374 are the only Series 90-30 CPUs that support Ethernet Global Data (EGD).

The Series 90-20 PLC provides a cost-effective platform for low I/O count applications. The primary objectives of the Series 90-20 PLC are as follows:

- To provide a small PLC that is easy to use, install, upgrade, and maintain.
- To provide a cost-effective family-compatible PLC.
- To provide easier system integration through standard communication hardware and protocols.

The Series 90 Micro PLC also provides a cost-effective platform for lower I/O count applications. The primary objectives of the Micro PLC are the same as those for the Series 90-20. In addition, the Micro offers the following:

- The Micro PLC has the CPU, power supply, inputs and outputs all built into one compact device.
- Most models also have a high speed counter.
- Because the CPU, power supply, and inputs and outputs are all built into one device, it is very easy to configure.

Note

For additional information, see the appendices in the back of this manual.

- Appendix A lists the memory size in bytes and the execution time in microseconds for each programming instruction.
- Appendix B describes how to interpret the message structure format when reading the PLC and I/O fault tables.
- Appendix C lists instruction mnemonics for searching or editing a program.
- Appendix D lists the special keyboard assignments used in the Logicmaster 90-30/20/Micro Software.
- Appendix E describes the use of floating-point math operations.

Note to Windows-Based PLC Programming Software Users

This manual was written for Logicmaster (a DOS-based PLC programming software) users. The Windows-based PLC software products, such as CIMPLICITY® Machine Edition Logic Developer and VersaPro®, provide PLC instruction set information in the software's built-in on-line help system rather than in a manual. Users of the Windows-based programming software should be aware that instructions appear differently from the way they appear on a Logicmaster screen (they still work the same in the PLC). The online help system has the most accurate information about using the instruction set in the Windows-based programming software. For a summary of major differences between the two software types, refer to Appendix F.

Chapter 2 System Operation

This chapter describes certain system operations of the Series 90-30, 90-20, and Micro PLC systems. These system operations include:

Section 1: PLC Sweep Summary

The logic program in the Series 90-30, 90-20, and Micro PLCs executes repeatedly until stopped by a command from the programmer or a command from another device. The sequence of operations necessary to execute a program one time is called a sweep. In addition to executing the logic program, the sweep includes obtaining data from input devices, sending data to output devices, performing internal housekeeping, servicing the programmer, and servicing other communications.

Series 90-30, 90-20, and Micro PLCs normally operate in **STANDARD PROGRAM SWEEP** mode. Other operating modes include **STOP WITH I/O DISABLED** mode, **STOP WITH I/O ENABLED** mode, and **CONSTANT SWEEP** mode. Each of these modes, described in this chapter, is controlled by external events and application configuration settings. The PLC makes the decision regarding its operating mode at the start of every sweep.

Standard Program Sweep

STANDARD PROGRAM SWEEP mode normally runs under all conditions. The CPU operates by executing an application program, updating I/O, and performing communications and other tasks. This occurs in a repetitive cycle called the CPU sweep. There are seven parts to the execution sequence of the Standard Program Sweep:

1. Start-of-sweep housekeeping
2. Input scan (read inputs)
3. Application program logic solution
4. Output scan (update outputs)
5. Programmer communications
6. System communications
7. Diagnostics

All of these steps execute every sweep. Although the Programmer Communications Window opens each sweep, programmer services only occur if a board fault has been detected or if the programming device issues a service request; that is, the Programmer Communications Window first checks for work to do and exits if there is none. The sequence of the standard program sweep is shown in the following figure.

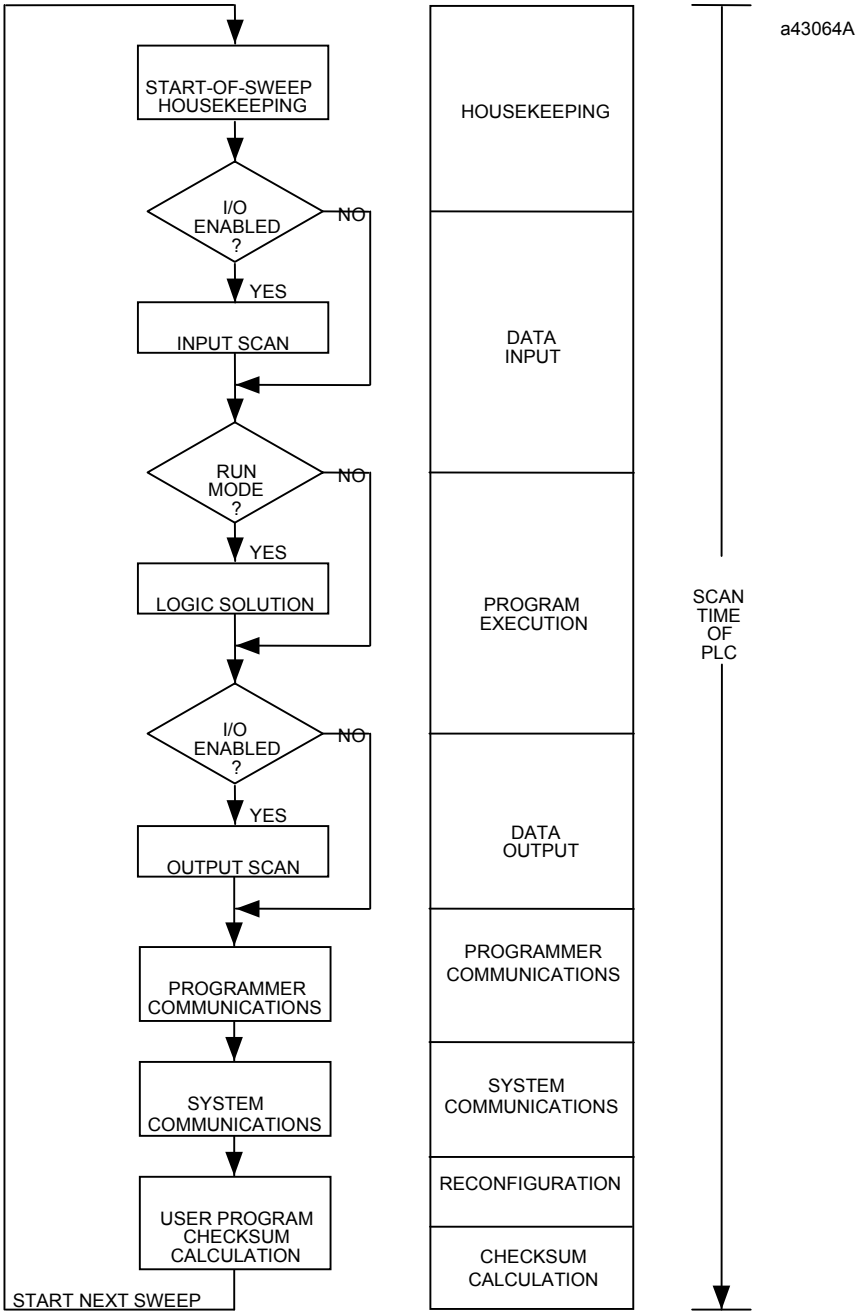


Figure 2-1. PLC Sweep

As shown in the PLC sweep sequence, several items are included in the sweep. These items contribute to the total sweep time as shown in the following table.

Table 2-1. Sweep Time Contribution

Sweep Element	Description		Time Contribution (milliseconds) [4]						
			Micro	211	311/313	331	34x	35x/36x	37x
Housekeeping	• Calculate sweep time • Schedule start of next sweep • Determine mode of next sweep • Update fault reference tables • Reset watchdog timer		0.368	0.898	0.714	0.705	0.424	0.279	0.027
Data Input	Input data is received from input and option modules		Note 5	See Tables 2-2 and 2-3 for scan time contributions					
Program Execution	User logic is solved		Execution time is dependent upon the length of the program and the types of instructions used in the program. Instruction execution times are listed in Appendix A.						
Data Output	Output data is sent to output and option modules		1.656	See Tables 2-2 and 2-3 for scan time contributions					
Programmer and System Communications	Service requests from programming devices and intelligent modules are processed[1]	HHP	1.93	6.526	4.426	4.524	2.476	0.334	N/A
		Programmer	0.380	3.536	2.383	2.454	1.248	0.517	0.026
		PCM[2]	N/A	N/A	N/A	3.337	1.943	0.482	0.029
Reconfiguration	Slots with faulted modules and empty slots are monitored		N/A[6]	N/A	0.458	0.639	0.463	0.319	0.243
Diagnostics	Verify user program integrity (time contribution is the time required per word checksummed each sweep) [3]		N/A[7]	0.083	0.050	0.048	0.031	0.010	0.022

1. The scan time contribution of external device service is dependent upon the mode of the communications window in which the service is processed. If the window mode is LIMITED, a maximum of 8 milliseconds for the 311, 313, 323, and 331 CPUs and 6 milliseconds for the 340 and higher CPUs will be spent during that window. If the window mode is **RUN-TO-COMPLETION**, a maximum of 50 milliseconds can be spent in that window, depending upon the number of requests which are presented simultaneously.
2. These measurements were taken with the PCM physically present but not configured and with no application task running on the PCM.
3. The number of words checksummed each sweep can be changed with the SVCREQ function block.
4. These measurements were taken with an empty program and the default configuration. The Series 90-30 PLCs were in an empty 10-slot rack with no extension racks connected. Also, the times in this table assume that there is no periodic subroutine active; the times will be longer if a periodic subroutine is active.
5. The data input time for the Micro PLC can be determined as follows: 0.365ms (fixed scan) + 0.036ms (filter time) x (total sweep time) / 0.5ms.
6. Since the Micro PLC has a static set of I/O, reconfiguration is not necessary.
7. Since the user program for the Micro PLC is in Flash memory, it will not be checked for integrity.

Table 2-2. I/O Scan Time Contributions (in milliseconds) for Series 90-30 35x, 36x and 37x CPUs

Module Type		35x and 36x Series CPUs			37x Series CPUs		
		Main Rack	Expansion Rack	Remote Rack	Main Rack	Expansion Rack	Remote Rack
8-point discrete input		.030	.055	.206	.030	.055	.206
16-point discrete input		.030	.055	.206	.030	.055	.206
32-point discrete input		.043	.073	.269	.048	.075	.272
8-point discrete output		.030	.053	.197	.024	.052	.198
16-point discrete output		.030	.053	.197	.030	.052	.199
32-point discrete output		.042	.070	.259	.047	.069	.258
Combination discrete input/output		.060	.112	.405	.052	.110	.408
4-channel analog input		.075	.105	.396	.085	.109	.403
2-channel analog output		.058	.114	.402	.046	.101	.393
16-channel analog input (current or voltage)		.978	1.446	3.999	.423	.700	1.741
8-channel analog output		1.274	1.988	4.472	.873	1.492	3.635
Combination analog input/output		1.220	1.999	4.338	.862	1.487	4.103
High Speed Counter		1.381	2.106	5.221	1.142	1.808	5.234
I/O Processor		1.574	2.402	6.388	1.270	2.125	6.269
Ethernet Interface (no connection)		.7129	2.067	3.681	.426	.795	2.302
Power Mate APM (1-axis)		1.527	2.581	6.388	1.236	2.073	6.032
Power Mate APM (2-axis)		1.807	2.864	7.805	1.539	2.439	7.369
DSM 302 *	40 AI, 6 AQ	2.143	3.315	9.527	1.801	2.963	9.275
	50AI, 9 AQ	2.427	3.732	11.092	2.075	3.373	10.840
	64 AI, 12 AQ	2.864	4.317	13.138	2.441	3.931	12.881
DSM314 *	1 Axis Configured	1.6	2.6	6.9	1.330	2.337	6.905
	2 Axes Configured	2.2	3.8	9.9	1.888	3.148	9.917
	3 Axes Configured	2.8	4.3	13.0	2.421	3.953	12.929
	4 Axes Configured	3.3	5.2	15.9	2.969	4.761	15.982
GCM	8 32-bit devices	8.826	16.932	21.179	7.386	9.520	20.591
GCM+	no devices	.567	.866	1.830	.457	.759	1.743
	32 64-word devices	19.497	25.588	80.871	17.036	24.390	80.044
GBC	no devices	.798	1.202	2.540	.544	.908	2.209
	16 64-word devices	29.976	40.570	131.702	26.976	38.564	130.639
PCM 311	not configured, or no application task	.476	N/A	N/A	.195	N/A	N/A
	running 20Kb application program	1.746	N/A	N/A	.538	N/A	N/A
ADC (no task)		.476	N/A	N/A	.193	N/A	N/A
I/O Link Master	no devices	.569	.865	1.932	.996	1.618	3.749
	sixteen 64-point devices	4.948	7.003	19.908	5.924	8.240	26.637
I/O Link Slave	32-point	.087	.146	.553	.095	.149	.540
	64-point	.154	.213	.789	.165	.219	.803

* For applications where the DSM's contributions to scan time will affect machine operation you may need to use the Do I/O function block, and the Suspend I/O and Fast Backplane Status Access service requests to transfer necessary data to and from the Motion module without getting all the data every scan. For the DSM302, refer to the *Motion Mate DSM302 for Series 90-30 PLCs User's Manual*, GFK1464 for details. For the DSM314, refer to the *Motion Mate DSM314 for Series 90-30 PLCs User's Manual*, GFK1742 for details. NOTE: The DSM314 will only work with the CPUs 350, 352, 360, 363, 364, and 374 and only with CPU firmware version 10.00 or later.

Table 2-3. I/O Scan Time Contributions (in milliseconds) for the Series 90-30 CPU311 through CPU341

Module Type		CPU Model						
			331			340/341		
		311/313 /323	Main Rack	Expansion Rack	Remote Rack	Main Rack	Expansion Rack	Remote Rack
8-point discrete input		.076	.054	.095	.255	.048	.089	.249
16-point discrete input		.075	.055	.097	.257	.048	.091	.250
32-point discrete input		.094	.094	.126	.335	.073	.115	.321
8-point discrete output		.084	.059	.097	.252	.053	.090	.246
16-point discrete output		.083	.061	.097	.253	.054	.090	.248
32-point discrete output		.109	.075	.129	.333	.079	.114	.320
8-point combination input/output		.165	.141	.218	.529	.098	.176	.489
4-channel analog input		.151	.132	.183	.490	.117	.160	.462
2-channel analog output		.161	.138	.182	.428	.099	.148	.392
High-Speed Counter		2.070	2.190	2.868	5.587	1.580	2.175	4.897
Power Mate APM (1-axis)		2.330	2.460	3.175	6.647	1.750	2.506	5.899
Power Mate APM (2-axis)		3.181	3.647	4.497	9.303	2.154	3.097	7.729
DSM 302*	40 AI, 6 AQ	3.613	4.081	5.239	11.430	2.552	3.648	9.697
	50AI, 9 AQ	4.127	4.611	5.899	13.310	2.911	4.170	11.406
	64 AI, 12 AQ	4.715	5.276	6.759	15.747	3.354	4.840	13.615
GCM	no devices	.041	.054	.063	.128	.038	.048	.085
	8 64-point devices	11.420	11.570	13.247	21.288	9.536	10.648	19.485
GCM+	no devices	.887	.967	1.164	1.920	.666	.901	1.626
	32 64-point devices	4.120	6.250	8.529	21.352	5.043	7.146	20.052
PCM 311	not configured, or no application task	N/A	3.350	N/A	N/A	1.684	N/A	N/A
	read 128 %R as fast as possible	N/A	4.900	N/A	N/A	2.052	N/A	N/A
ADC 311		N/A	3.340	N/A	N/A	1.678	N/A	N/A
16-channel analog input (current or voltage)		1.370	1.450	1.937	4.186	1.092	1.570	3.796
I/O Link Master	no devices	1.910	2.030	1.169	1.925	.678	.904	1.628
	sixteen 64-point devices	6.020	6.170	8.399	21.291	4.992	6.985	20.010
I/O Link Slave	32-point	.206	.222	.289	.689	.146	.226	.636
	64-point	.331	.350	.409	1.009	.244	.321	.926

* For applications where the DSM's contributions to scan time will affect machine operation you may need to use the Do I/O function block, and the Suspend I/O and Fast Backplane Status Access service requests to transfer necessary data to and from the Motion module without getting all the data every scan. Refer to the *Motion Mate DSM302 for Series 90-30 PLCs User's Manual*, GFK1464 for details. NOTE: The DSM314 is not supported by the 311 through 341 CPUs.

Sweep Time Calculation

Table 2-1 lists the seven items that contribute to the sweep time of the PLC. The sweep time consists of fixed times (housekeeping and diagnostics) and variable times. Variable times vary according to the I/O configuration, size of the user program, and the type of programming device connected to the PLC.

Example of Sweep Time Calculation

An example of the calculations for determining the sweep time for a Series 90-30 model 331 PLC are shown in the table below.

The modules and instructions used for these calculations are listed below:

- Input modules: five 16-point Series 90-30 input modules.
- Output modules: four 16-point Series 90-30 output modules.
- Programming instructions: A 1200-step program consisting of 700 Boolean instructions (LD, AND, OR, etc.), 300 output coils (OUT, OUTM, etc.), and 200 math functions (ADD, SUB, etc.).

Sweep Component	Calculation	Time Contribution		
		Without Programmer	With HHP	With Logicmaster
Housekeeping	0.705ms	0.705ms	0.705ms	0.705ms
Data Input	0.055 x 5 = 0.275ms	0.275ms	0.275ms	0.275ms
Program Execution	1000 x 0.4µs* + 200 x 89µs** + 18.2ms	18.2ms	18.2ms	18.2ms
Data Output	0.061 x 4 = 0.244ms	0.244ms	0.244ms	0.244ms
Programmer Service	0.4ms + programmer time + 0.6ms	0ms	4.524ms	2.454ms
Non-Programmer Service	None in this example	0ms	0ms	0ms
Reconfiguration	0.639ms	0.639ms	0.639ms	0.639ms
Diagnostics	0.048ms	0.048ms	0.048ms	0.048ms
PLC Sweep Time	Housekeeping + Data Input + Program Execution + Data Output + Programmer Service + Non-Programmer Service + Diagnostics	12.611ms	17.135ms	15.065ms

PLC Sweep Details

This section discusses details of the major portions of the PLC Sweep:

1. Housekeeping
2. Input Scan
3. Application Program Logic Scan
4. Output Scan
5. Programmer Service
6. System Communications
7. Reconfiguration
8. Checksum Calculation

1. Housekeeping

The housekeeping portion of the sweep performs all of the tasks necessary to prepare for the start of the sweep. If the PLC is in **CONSTANT SWEEP** mode, the sweep is delayed until the required sweep time elapses. If the required time has already elapsed, the OV_SWP %SA0002 contact is set, and the sweep continues without delay. Next, timer values (hundredths, tenths, and seconds) are updated by calculating the difference from the start of the previous sweep and the new sweep time. In order to maintain accuracy, the actual start of sweep is recorded in 100 microsecond increments. Each timer has a remainder field which contains the number of 100 microsecond increments that have occurred since the last time the timer value was incremented.

2. Input Scan

Scanning of inputs occurs during the input scan portion of the sweep, just prior to the logic solution. During this part of the sweep, all Series 90-30 input modules are scanned and their data stored in %I (discrete inputs) or %AI (analog inputs) memory, as appropriate. Any global data input received by a Genius Communications Module (GCM), an Enhanced Genius Communications Module (GCM+), or a Genius Bus Controller (GBC) is stored in %G memory.

Modules are scanned in ascending reference address order, starting with any installed Genius Module, then discrete input modules, and finally analog input modules.

If the CPU is in **STOP** mode and the CPU is configured to not scan I/O in **STOP** mode, the input scan is skipped.

3. Application Program Logic Scan or Solution

The application program logic scan occurs immediately following the completion of the input scan. The application program logic scan performs two main tasks: (1) solving/executing the program logic and (2) updating %Q, %AI, and %AQ output memory. (Output modules, however, are not updated until the output scan occurs). In general, ladder logic is solved from left to right and top to bottom, although this flow direction can be altered temporarily by subroutine calls and jumps. The

logic solution ends when an END instruction is encountered or when the default END OF PROGRAM LOGIC is reached.

The 313 and higher CPUs have an Instruction Sequence Coprocessor (ISCP) that executes the Boolean instructions, and an 80C188,80386 or AMD SC 520 microprocessor executes the timer, counter, and function blocks. In the Model 311 and 90-20 CPUs, the 80C188 executes all Boolean, timer, counter, and function block instructions. On the Micro, the H8 processor executes all Boolean and function blocks.

A list of execution times for each programming function can be found in Appendix A.

4. Output Scan

Outputs are scanned during the output scan portion of the sweep, immediately following the logic solution. Outputs are updated using data from %Q (for discrete outputs) and %AQ (for analog outputs) memory, as appropriate. If you have a Genius Communications Module or Genius Bus Controller that is configured to transmit global data, then data from %G memory is sent to the GCM, GCM+, or GBC. The Series 90-20 and Micro output scans include discrete outputs only.

During the output scan, all Series 90-30 output modules are scanned in ascending reference address order. The output scan is completed when all output data has been sent to all Series 90-30 output modules.

If the CPU is in the **`STOP`** mode and ***IPScan-Stop*** parameter on the CPU configuration screen is set to ***NO***, the output scan is skipped.

Caution

If the *IPScan-Stop* parameter on the CPU configuration screen is set to *YES*, real-world outputs may be turned ON even when the PLC is in STOP mode, because the PLC will write the current values in the output tables to the output modules during the Output Scan.

5. Programmer Communications Window

This part of the sweep is dedicated to communicating with the programmer. If there is a programmer attached, the CPU executes the programmer communications window. The programmer communications window will not execute if there is no programmer attached and no module to be configured in the system. Only one module is configured each sweep.

Support is provided for the Hand-Held Programmer and for other programmers that can connect to the serial port and use the Series Ninety Protocol (SNP) protocol. Support is also provided for programmer communications with intelligent option modules.

Programmer Communications Window Modes

- **Limited Mode**. In the default Limited window mode, the CPU performs one operation for the programmer each sweep, that is, it honors one service request or response to one key press. If the programmer makes a request that requires more than 6 (or 8 depending on the CPU—see Note) milliseconds to process, the request processing is spread out over several

sweeps so that no sweep is impacted by more than 6 (or 8 depending on the CPU—see Note) milliseconds.

Note

The time limit for the communications window is 6 milliseconds for the 340 and higher CPUs and 8 milliseconds for the 311, 313, 323, and 331 models.

- **Complete Mode.** In the Complete mode, the CPU will conduct programmer communications until they are complete or until 50 milliseconds elapses.

The following figure is a flow chart for the programmer communications portion of the sweep.

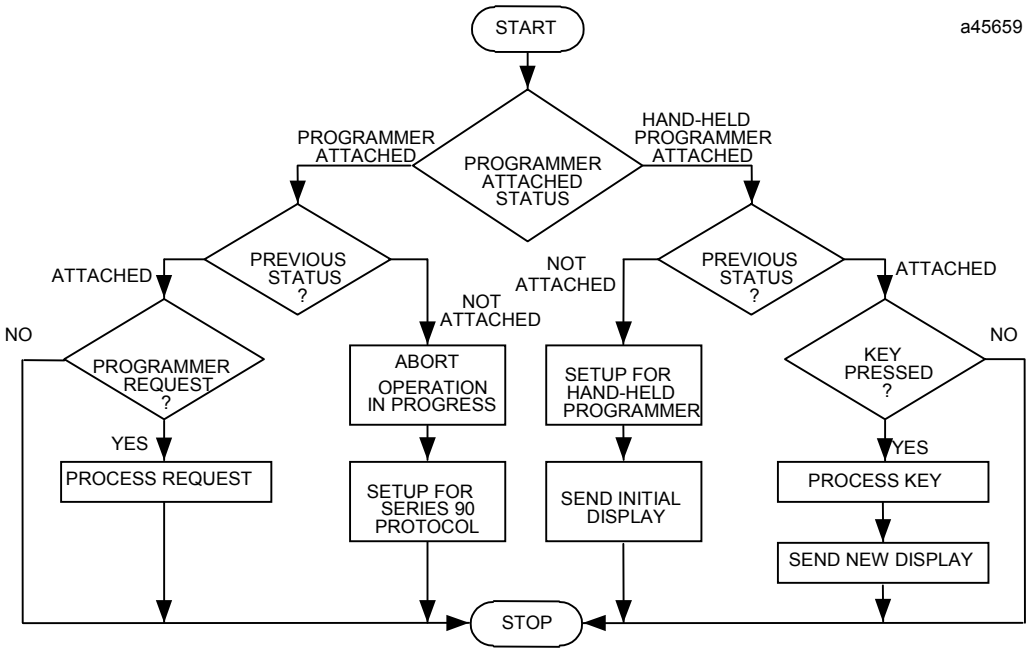


Figure 2-2. Programmer Communications Window Flow Chart

6. System Communications Window (Models 331 and Higher)

This is the part of the sweep where communications requests from intelligent option modules, such as the PCM or DSM, are processed (see flow chart). Requests are serviced on a first-come-first-served basis. However, since intelligent option modules are polled in a round-robin fashion, no intelligent option module has priority over any other intelligent option module.

In the default **`Run-to-Completion`** mode, the length of the system communications window is limited to 50 milliseconds. If an intelligent option module makes a request that requires more than 50 milliseconds to process, the request is spread out over multiple sweeps so that no one sweep is impacted by more than 50 milliseconds.

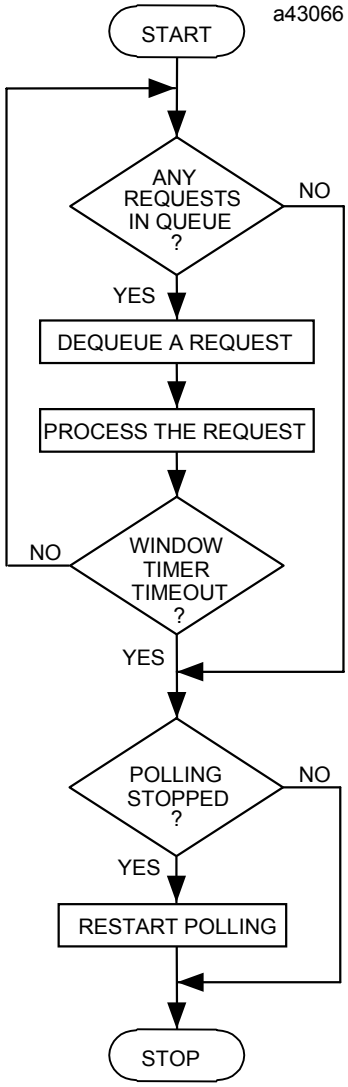


Figure 2-3. System Communications Window Flow Chart

7. Reconfiguration

During this portion of the sweep, the CPU checks the actual hardware lineup against the configured hardware lineup. Slots that are configured for a module but that are empty physically, or slots that contain faulted modules, will not be scanned by the CPU (i.e. the CPU will not read any input data from, and will not send any output data to that module or slot). During reconfiguration, if the CPU detects that a slot previously identified as containing a faulted module now has a good module, or that a configured module has been physically added to the PLC, it will begin scanning that module.

Reconfiguration enables the CPU to do the following:

- Recognize a legitimate change that you make in the configuration.
- Ignore potentially corrupted or inaccurate input data from faulted or missing modules.
- Avoid sending output data that could become corrupted by a faulted output module.

8. Checksum Calculation

A checksum calculation is performed on the user program at the end of every sweep. Since it would take too long to calculate the checksum of the entire program, you can specify the number of words from 0 to 32 to be checked on the CPU configuration screen.

If the calculated checksum does not match the reference checksum, the program checksum failure exception flag is raised. This causes a fault entry to be inserted into the PLC fault table and the PLC mode to be changed to **STOP**. If the checksum calculation fails, the programmer communications window is not affected. The default number of words to be checksummed is 8.

PCM Communications with the PLC (Models 331 and Higher)

There is no way for intelligent option modules (IOM), such as the PCM, to interrupt the CPU when they need service. The CPU must poll (check periodically) each intelligent option module for service requests. This polling occurs asynchronously in the background during the sweep (see flow chart below).

When an intelligent option module is polled and sends the CPU a service request, the request is queued for processing during the system communications window.

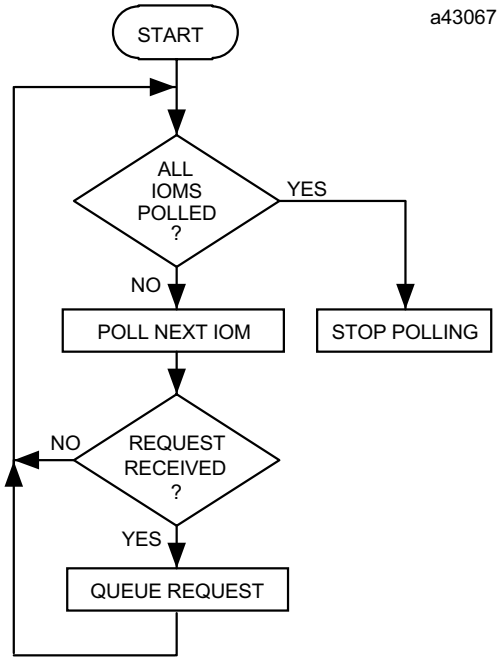


Figure 2-4. PCM Communications with the PLC

Digital Servo Module (DSM) Communications with the PLC

The DSM302 and DSM314 are intelligent option modules that operate asynchronously with the Series 90-30 CPU module. Data is exchanged between the CPU and a DSM automatically via %Q, %I, %AQ, and %AI memory. A PLC CPU requires time to read and write the exchange data across the PLC backplane with the DSM module. Table 2-2 lists the sweep impact for the various possible DSM configurations. For additional timing considerations that apply to the DSM modules, refer to the following manuals:

- *Motion Mate DSM302 for Series 90-30 PLCs User's Manual*, GFK-1464.
- *Motion Mate DSM314 for Series 90-30 PLCs User's Manual*, GFK-1742.

Standard Program Sweep Variations

In addition to the normal execution of the standard program sweep, certain variations can be encountered or forced. These variations, described in the following paragraphs, can be displayed and/or changed from the programming software.

Constant Sweep Time Mode

In the standard program sweep, each sweep executes as quickly as possible with a varying amount of time consumed each sweep. An alternative to this is **CONSTANT SWEEP TIME** mode, where each sweep consumes the same amount of time. You can achieve this by setting the Configured Constant Sweep, which will then become the default sweep mode, thereby taking effect each time the PLC goes from **STOP** to **RUN** mode. You may set a **CONSTANT SWEEP TIME** mode value between 5 to 200 milliseconds for CPUs 311-341 or between 5 and 500 milliseconds for the 350-364 and 374 CPUs.

Due to variations in the time required for various parts of the PLC sweep, the constant sweep time should be set at least 10 milliseconds higher than the sweep time that is displayed on the status line when the PLC is in **NORMAL SWEEP** mode. This prevents the occurrence of extraneous oversweep faults.

Use the **CONSTANT SWEEP TIME** mode when I/O points or register values must be polled at a constant frequency, such as in control algorithms. Another reason might be to ensure that a certain amount of time elapses between the output scan and the next sweep's input scan, permitting inputs to settle after receiving output data from the program.

If the constant sweep timer expires before the sweep completes, the entire sweep, including the communications windows, is completed. However, an oversweep fault is logged at the beginning of the next sweep.

Configuring Constant Sweep Mode

There are two ways to configure Constant Sweep Mode:

- In Logicmaster configuration software, the CPU configuration screen has configurable Sweep Mode and Sweep Timer parameters. After making your selections, you must store the configuration from the programmer to the PLC during **STOP** mode before the changes will take effect. Once stored, this configuration becomes the default sweep mode.

- In Logicmaster programming software, the PLC Sweep Table selection on the PLC Control and Status menu has Sweep Mode and Timing parameter selection options. The parameters on this screen can only be edited in **RUN** mode. Changes made from this screen are only stored to the PLC, not to the folder on your PC, and are only effective while the PLC remains in Run mode. Once the PLC stops, it assumes the default Sweep Mode, which becomes effective the next time the PLC goes into Run mode. This method for temporarily configuring the Sweep Mode is useful for system design and debug operations.

PLC Sweep When in STOP Mode

When the PLC is in **STOP** mode, the application program is not executed. Communications with the programmer and intelligent option modules continue. In addition, faulted module polling and module reconfiguration execution continue while in **STOP** mode. For efficiency, the operating system uses larger "time-slice" values than those used in **RUN** mode (usually about 50 milliseconds per window). You can choose whether or not the I/O is scanned. I/O scans may execute in **STOP** mode if the ***IOScan-Stop*** parameter on the CPU detail screen is set to **YES**.

Caution

If the *IPScan-Stop* parameter on the CPU detail screen is set to *YES*, real-world outputs may be turned ON even when the PLC is in STOP mode, because the PLC will write the current values in the output tables to the output modules during the Output Scan.

Communication Window Modes

The default window mode for the programmer communication window is "Limited" mode. That means that if a request takes more than 6 milliseconds to process, it is processed over multiple sweeps, so that no one sweep is impacted by more than 6 milliseconds. For the 313, 323, and 331 CPUs, the sweep impact may be as much as 12 milliseconds during a **RUN**-mode store. The active window mode can be changed using the "Sweep Control" screen in Logicmaster—for instructions on changing the active window mode, refer to Chapter 5, "PLC Control and Status," in the *Logicmaster 90™ Series 90™-30/20/Micro Programming Software User's Manual* (GFK-0466).

Note

If the system window mode is changed to Limited, then option modules such as the PCM or GBC that communicate with the PLC using the system window will have less impact on sweep time, but response to their requests will be slower.

Keylock Switch on 35x, 36x and 37x Series CPUs: Change Mode and Flash Protect

All 350—374 CPUs have a keylock switch (CPUs 311-341 do not); however, some versions of CPU firmware do not support all keylock switch features. These differences are discussed in this section. Note that the keylock switches on some of these CPUs are labeled ON/RUN and OFF/STOP, and on others are labeled ON and OFF. Regardless of the labeling, all of these keylock switches work as described below.

Flash Memory Protection (Hard-Wired)

This hard-wired, non-configurable feature can be used to prevent Flash memory from being changed by unauthorized people (people without a key). When the keylock switch is in the ON position, Flash memory cannot be written to. Flash memory can only be written to when this switch is OFF. This keylock switch feature is always in effect, regardless of how the next two configurable features are set.

Run/Stop (Configurable)

This configurable feature was introduced in CPU firmware release 7.00. It is set by the **R/S Switch** parameter on the CPU configuration screen. The **R/S Switch** parameter is set to *Disabled* by default. If the **R/S Switch** parameter is set to *Enabled,* you can stop the PLC by turning the keylock switch to OFF, and start the PLC by turning the switch to ON (if there are no faults). If faults exist, one of the following will happen:

- **If the PLC has a non-fatal fault**, turning the keylock switch from OFF to ON will cause the PLC to go into run mode, and the RUN light will turn on steady, but the fault tables will not be cleared.
- **If the PLC has a fatal fault,** turning the keylock switch from OFF to ON will cause the RUN light to flash on and off for a period of five seconds, and the PLC will not go into run mode. This flashing light indicates the presence of one or more fatal faults in the Fault Tables. You can try to clear the fault table faults by turning the keylock switch from OFF to ON again during the five-second period. (If the five-second period has expired, turning the keylock switch from OFF to ON will start another five-second period.) If the faults do not clear using this method, you will have to remedy the causes of the fatal faults before being able to resume operation. See Chapter 3 for fault details.

Other Run/Stop Keylock Switch Considerations

- If the **R/S Switch** parameter is set to *Enabled* and the keylock switch is in the OFF position, the PLC will be in STOP mode, and the programming software cannot be used to place the PLC into RUN mode.
- If the **R/S Switch** parameter is set to *Enabled*, the keylock switch is in the ON position, and there are no fatal faults, the programming software can be used to toggle the PLC between the RUN and STOP modes.

- If the **R/S Switch** parameter is set to *Enabled*, the keylock switch is in the ON position, but the PLC is stopped, you can place the PLC into RUN mode by either turning the keylock switch to the OFF position and then back to ON, or by using the programming software.

RAM Memory and Override Protection (Configurable)

This feature was introduced in CPU firmware release 8.00. It is set by the **Mem Protect** parameter on the CPU configuration screen. The **Mem Protect** parameter is set to *Disabled* by default.

If the **Mem. Protect** parameter is set to *Enabled*, and the keylock switch is in the ON position, the following is true:

- User RAM memory (program and configuration) cannot be changed.
- Discrete points cannot be overridden.
- The Time of Day (TOD) clock cannot be changed with the Hand-Held Programmer (however, the TOD clock can still be changed using the configuration software).

Safeguard your Keys

Each new 350—374 CPU is shipped with two keys for the keylock switch. If you use one or more of the keylock switch protection features described above, we recommend you carefully safeguard your keys. If they are lost, misplaced, or stolen, you may be locked out from working on your PLC, and unauthorized persons may have access to it. You may want to purchase spare keys for backup purposes, or if more than two persons need access to the PLC. A keylock switch key kit, containing three sets of keys, can be purchased from a GE Fanuc distributor. When ordering, request catalog number 44A736756-G01. All 350—374 CPUs use the same key.

Disabling Keylock Switch Features

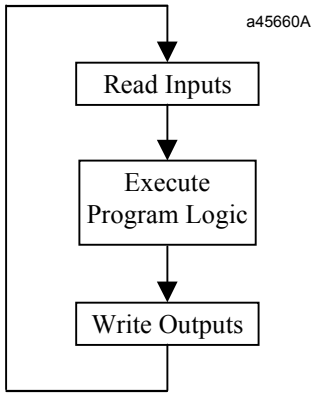
If you do not need to use any of the protection features of the keylock switch, you can choose to disable them all. To do so, leave the keylock switch set to the OFF position, and set the **R/S Switch** and **Mem. Protect** parameters (described above) to *Disabled* (their default setting). In this condition, all keylock switch protection features will be disabled, and you will not need to use a key to access the PLC.

Section 2: Program Organization and User References/Data

The user memory size for the Series 90-30 programmable controllers is listed in the following table.

User Memory Size	
CPU Models	**User Memory (Kbytes)**
CPU311	6
CPU313, CPU323	12
CPU331	16
CPU340	32
CPU341	80
CPU350	80 (release 9.00 and later) 32 (prior to release 9.00)
CPU351, CPU352, CPU360, CPU363, CPU364, CPU374	240 (release 9.00 and later) 80 (prior to release 9.00)

Beginning with firmware release 9.00 CPUs, %R, %AI, and %AQ memory sizes for the 351, 352, 360, 363, 364 and 374 CPUs are configurable. (For details, refer to the *Logicmaster 90™ Series 90™-30/20/Micro Programming Software User's Manual*, GFK-0466K or later or the User's Manual for your programmer software). A program for the Series 90-20 programmable controller can be up to 2 KB in size for a Model 211 CPU, and the maximum number of rungs allowed per logic block (main or subroutine) is 3000. For Series 90-30 PLCs, the maximum block size is 80 kilobytes for C blocks and 16 kilobytes for LD and SFC blocks; however, in an SFC block, some of the 16 KB is used for the internal data block. As shown in the next figure, user program logic is executed repeatedly by the PLC while the PLC is in normal Run mode.



Refer to the *Series 90-30 Programmable Controller Installation and Hardware Manual,* GFK-0356, or the *Series 90-20 Programmable Controller User's Manual*, GFK-0551, for a listing of program sizes and reference limits for each model CPU.

All programs have a variable table that lists the variable and reference descriptions that have been assigned in the user program.

The block declaration editor lists subroutine blocks declared in the main program.

Subroutine Blocks

A program can "call" subroutine blocks as it executes. A subroutine must be declared through the block declaration editor before a CALL instruction can be used for that subroutine. A maximum of 64 subroutine block declarations in the program and 64 CALL instructions are allowed for each logic block in the program. The maximum size of a subroutine block is 16 KB or 3000 rungs, but the main program and all subroutines must fit within the logic size constraints for that CPU model.

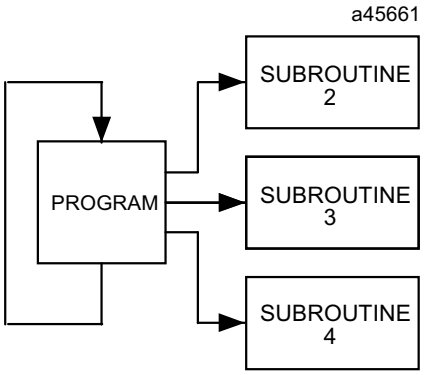
Note

Subroutine blocks are not supported in the Series 90-20 PLC or the Micro PLC.

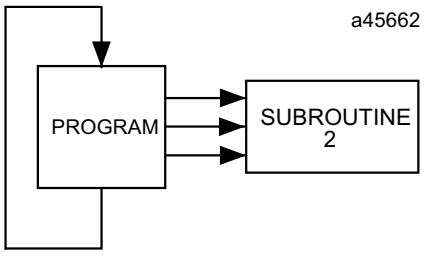
The use of subroutines is optional. Dividing a program into smaller subroutines can simplify programming, enhance understanding of the control algorithm, and possibly reduce the overall amount of logic needed for the program.

Examples of Using Subroutine Blocks

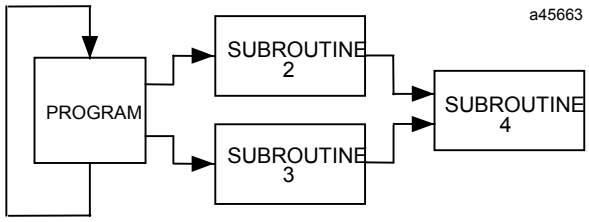
As an example, the logic for a program could be divided into three subroutines, each of which could be called as needed from the program. In this example, the program block might contain little logic, serving primarily to sequence the subroutine blocks.



A subroutine block can be used many times as the program executes. Logic which needs to be repeated several times in a program could be entered in a subroutine block. Calls would then be made to that subroutine block to access the logic. In this way, total program size is reduced.



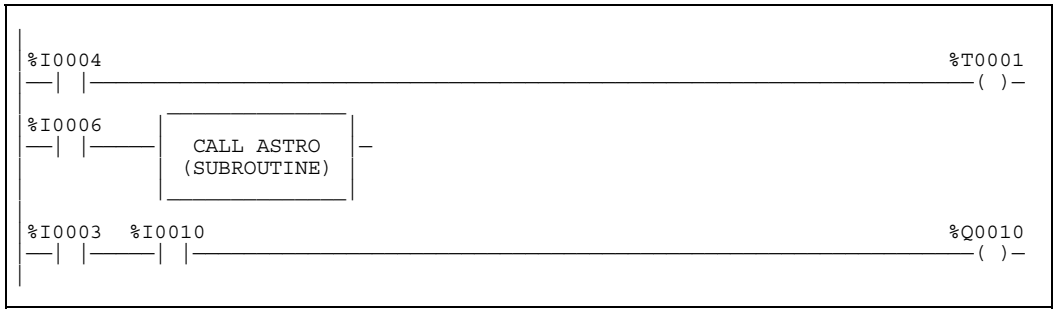
In addition to being called from the program, subroutine blocks can also be called by other subroutine blocks (this is called “nesting”). A subroutine block may even call itself.



The PLC will only allow eight nested calls before an “Application Stack Overflow” fault is logged and the PLC transitions to **`STOP/Fault`** mode. The call level nesting counts the main program as level 1.

How Blocks Are Called

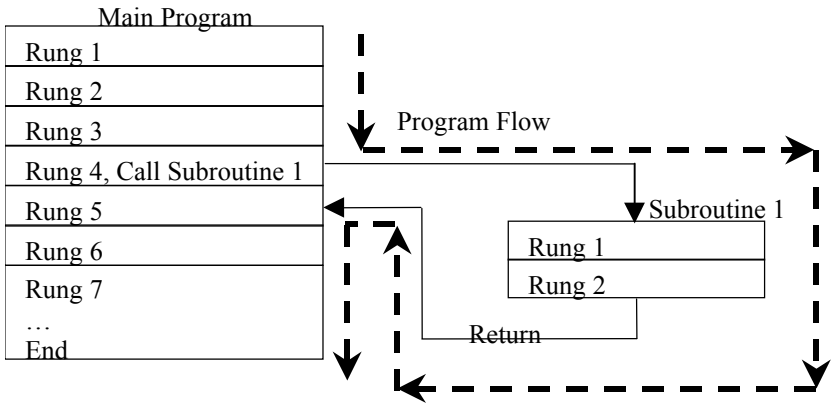
A subroutine block executes when called from program logic in a ladder program or from another subroutine block.



This example shows the subroutine CALL instruction as it will appear on the ladder logic screen.

Execution Sequence in Programs Containing Subroutines

If a subroutine is called from a program or other subroutine, the called subroutine will execute to its end, then return control back to the program or subroutine that called it. Control will return to the rung following the rung that contains the subroutine call. In the example below, the heavy dotted line shows program flow (the order in which logic is executed). In this example, a simple two-rung subroutine is called from Rung 4 of the Main Program. After the two subroutine rungs are executed, program flow returns to the Main Program, starting with Rung 5.



Periodic Subroutines

Version 4.20 or later of the 340 and higher CPUs support periodic subroutines. Please note the following restrictions:

1. Timer (TMR, ONDTR, and OFDTR) function blocks will not execute properly within a periodic subroutine. A DOIO function block within a periodic subroutine whose reference range includes references assigned to a Smart I/O Module (HSC, APM, DSM, Genius, etc.) will cause the CPU to lose communication with the module. The FST_SCN and LST_SCN contacts (%S1 and %S2) will have an indeterminate value during execution of the periodic subroutine. A periodic subroutine cannot call or be called by other subroutines.
2. The latency for the periodic subroutine (that is, the maximum interval between the time the periodic subroutine should have executed and the time it actually executes) can be around 0.35 milliseconds if there is no PCM, CMM, or ADC module in the main rack. If there is a PCM, CMM or ADC module in the main rack—even if it is not configured or used—the latency can be almost 2.25 milliseconds. For that reason, use of the periodic subroutine with PCM-based products is ***not*** recommended.

User References

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

Table 2-4. Register References

Type	Description
%R	The prefix %R is used 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 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 value.

Note

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

Table 2-5. Discrete References

Type	Description
%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 the configuration software or the Hand-Held Programmer. Until a reference address is assigned, no data will be received from the module. %I data can be retentive or non-retentive.
%Q	The %Q prefix represents physical output references. The coil check function of Logicmaster 90-30/20/Micro software checks for multiple uses of %Q references with relay coils or outputs on functions. Beginning with Release 3 of the software, you can select the level of coil checking desired (`SINGLE`, `WARN MULTIPLE`, or `MULTIPLE`). Refer to the Programming Software User's Manual, GFK-0466, for more information about this feature. 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 during the output scan. A reference address is assigned to discrete output modules using the configuration software or the Hand-Held Programmer. Until a reference address is assigned, no data is sent to the module. A particular %Q reference may be either retentive or non-retentive. *
%M	The %M prefix represents internal references. The coil check function checks for multiple uses of %M references with relay coils or outputs on functions. Beginning with Release 3 of the software, you can select the level of coil checking desired (`SINGLE`, `WARN MULTIPLE`, or `MULTIPLE`). Refer to GFK-0466 for more information about this feature. A particular %M reference may be either retentive or non-retentive. *
%T	The %T prefix represents temporary references. Because these references are never checked for multiple coil use, they can be used many times in the same program, even when coil use checking is enabled. %T can 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 not retained through power loss or **`RUN-TO-STOP-TO-RUN`** transitions and cannot be used with retentive coils.
%S	The %S prefix represents system status references. These references are used to access special PLC data, such as timers, scan information, and fault information. System references include %S, %SA, %SB, and %SC references. %S, %SA, %SB, and %SC can be used on any contacts. %SA, %SB, and %SC can be used on retentive coils –(M)–. %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.
%G	The %G prefix represents global data references. These references are used to access data shared among several PLCs. %G references can be used on contacts and retentive coils because %G memory is always retentive. %G cannot be used on non-retentive coils.

* Retentiveness is based on the type of coil. For more information, refer to "Retentiveness of Data" on the next page.

Nicknames

A user may, optionally, assign a nickname to a reference address. A nickname is useful because it can convey information to the user about the purpose or function of the address. For example, in a PLC system installed in a factory, output coil %Q0001 is used to energize a motor starter relay that controls a physical pump, commonly called "Pump Number 1" by the factory's employees. Assigning the nickname PUMP1 to %Q0001 would help an employee who is troubleshooting the system to recognize the purpose of %Q0001.

Nicknames must begin with a letter and may be from one to seven characters long. To distinguish between a memory address (reference) and a nickname, a percent sign (%) is used as the first character of a memory address. So, for example, M1 is considered by the PLC to be a nickname, but %M1 is considered to be a memory address. For more information about nicknames, please see manual GFK-0466 (the Logicmaster user's manual for the Series 90-30 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 and transitional coils. Note that counters do not use the same kind of transition bits as coils. Transition bits for counters are stored within the locating reference.

In the Model 331 and higher CPUs, override bits can be set. 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. CPU Models 323, 321, 313, and 311, and the Micro CPUs do not support overriding discrete references.

Retentiveness of Data

Data is said to be retentive if it is saved by the PLC when the PLC is stopped. The Series 90 PLC preserves program logic, fault tables and diagnostics, overrides and output forces, word data (%R, %AI, %AQ), bit data (%I, %SC, %G, fault bits and reserved bits), %Q and %M data (unless used with non-retentive coils), and word data stored in %Q and %M. %T data is not saved. Although, as stated above, %SC bit data is retentive, the defaults for %S, %SA, and %SB are non-retentive.

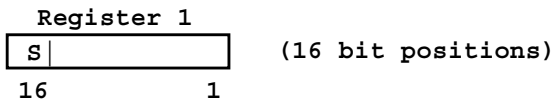
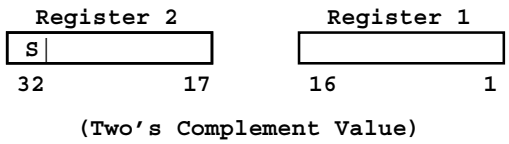
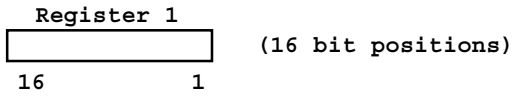
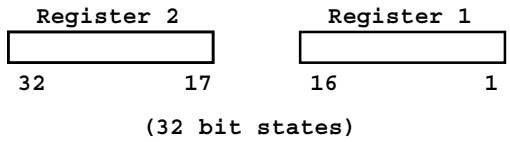
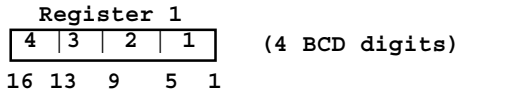
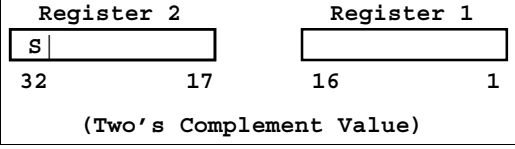
%Q and %M references are non-retentive (that is, cleared at power-up when the PLC switches from **STOP** to **RUN**) whenever they are used with non-retentive coils. Non-retentive coils include coils —( )—, negated coils —(/)—, SET coils —(S)—, and RESET coils —(R)—.

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 %Q0001 was last programmed as the reference of a retentive coil, the %Q0001 data will be retentive. However, if %Q0001 was last programmed on a non-retentive coil, the %Q0001 data will be non-retentive.

Data Types

Table 2-6. Data Types

Type	Name	Description	Data Format
INT	Signed Integer	Signed integers use 16-bit memory data locations, and are represented in 2's complement notation. (Bit 16 is the sign bit.) The valid range of an INT data type is –32,768 to +32,767.	
DINT	Double Precision Signed Integer	Double precision signed integers are stored in 32-bit data memory locations (actually two consecutive 16-bit memory locations) and represented in 2's complement notation. (Bit 32 is the sign bit.) The valid range of a DINT data type is –2,147,483,648 to +2,147,483,647.	
BIT	Bit	A Bit data type is the smallest unit of memory. It has two states, 1 or 0. A BIT string may have length N.	
BYTE	Byte	A Byte data type has an 8-bit value. The valid range is 0 to 255 (0 to FF in hexadecimal).	
WORD	Word	A Word data type uses 16 consecutive bits of data memory; but, instead of the bits in the data location representing a number, the bits are independent of each other. Each bit represents its own binary state (1 or 0), and the bits are not looked at together to represent an integer number. The valid range of word values is 0 to FFFF.	
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 16 bits. The valid range of double word values is 0 to FFFFFFFF.	
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.	
REAL	Floating Point	Real numbers use 32 consecutive bits (actually 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.	

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

System Status References

System status references in the Series 90 PLC are assigned to %S, %SA, %SB, and %SC memory. They each have a nickname. Examples of time tick references include T_10MS, T_100MS, T_SEC, and T_MIN. Examples of convenience 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 system status references that can be used in an application program. When entering logic, either the reference or the nickname can be used. Refer to chapter 3, "Fault Explanations and Correction," for more detailed fault descriptions and information on correcting the fault. You cannot use these special nicknames to name other memory references.

Table 2-7. System Status References

Reference	Nickname	Definition
%S0001	FST_SCN	Set to 1 when the current sweep is the first sweep.
%S0002	LST_SCN	Reset from 1 to 0 when the current sweep is the last sweep.
%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. 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. 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.
%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 reference is updated once per sweep.
%S0017	SNPXACT	SNP-X host is actively attached to the CPU.
%S0018	SNPX_RD	SNP-X host has read data from the CPU.
%S0019	SNPX_WT	SNP-X host has written data to the CPU.
%S0020		Set ON when a relational function using REAL data executes successfully. It is cleared when either input is NaN (Not a Number).
%S0032		Reserved for use by the programming software.
%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 discrete bit can be cleared by again storing the program to the CPU. If the fault was due to a hardware RAM failure, the CPU must be replaced.
%SA0002	OV_SWP	Set when the PLC detects that the previous sweep took longer than the time specified by the user. Cleared when the PLC detects that the previous sweep did not take longer than the specified time. It is also cleared during the transition from **`STOP`** to **`RUN`** mode. Only valid if the PLC is in **`CONSTANT SWEEP`** mode.

Reference	Nickname	Definition
%SA0003	APL_FLT	Set when an application fault occurs. Cleared when the PLC transitions from **STOP** to **RUN** mode.
%SA0009	CFG_MM	Set when a configuration mismatch is detected during system power-up or during a store of the configuration. Cleared by powering up the PLC when no mismatches are present or during a store of configuration that matches hardware.
%SA0010	HRD_CPU	Set when the diagnostics detects a problem with the CPU hardware. Cleared by replacing the CPU module.
%SA0011	LOW_BAT	Set when a low battery fault occurs. Cleared by replacing the battery and ensuring that the PLC powers up without the low battery condition.
%SA0014	LOS_IOM	Set when an I/O module stops communicating with the PLC CPU. Cleared by replacing the module and cycling power on the main rack.
%SA0015	LOS_SIO	Set when an option module stops communicating with the PLC CPU. Cleared by replacing the module and cycling power on the main rack.
%SA0019	ADD_IOM	Set when an I/O module is added to a rack. Cleared by cycling power on the main rack and when the configuration matches the hardware after a store.
%SA0020	ADD_SIO	Set when an option module is added to a rack. Cleared by cycling power on the main rack and when the configuration matches the hardware after a store.
%SA0027	HRD_SIO	Set when a hardware failure is detected in an option module. Cleared by replacing the module and cycling power on the main rack.
%SA0031	SFT_SIO	Set when an unrecoverable software fault is detected in an option module. Cleared by cycling power on the main rack and when the configuration matches the hardware.
%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.
%SB0011	BAD_PWD	Set when a password access violation occurs. Cleared when the PLC fault table is cleared.
%SB0013	SFT_CPU	Set when the CPU detects an unrecoverable error in the software. Cleared by clearing the PLC fault table.
%SB0014	STOR_ER	Set when an error occurs during a programmer store operation. Cleared when a store operation is completed successfully.
%SC0009	ANY_FLT	Set when any fault occurs. Cleared when both fault tables have no entries.
%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 has no entries.
%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 has no entries.
%SC0012	SY_PRES	Set as long as there is at least one entry in the PLC fault table. Cleared when the PLC fault table has no entries.
%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 has no entries.
%SC0014	HRD_FLT	Set when a hardware fault occurs. Cleared when both fault tables have no entries.
%SC0015	SFT_FLT	Set when a software fault occurs. Cleared when both fault tables have no entries.

Note: Any %S reference not listed here is reserved and must not be used in program logic.

Function Block Structure

Each rung of logic is composed of one or more programming instructions. These may be simple relays or more complex functions.

Format of Ladder Logic Relays

The programming software includes several types of relay functions. These functions provide basic flow and control of logic in the program. Examples include a normally open relay contact and a negated coil. Each of these relay contacts and coils has one input and one output. Together, they provide logic flow through the contact or coil.

Each relay contact or coil must be given a reference which is entered when selecting the relay. For a contact, the reference represents a location in memory that determines the flow of power into the contact. In the following example, if reference %I0122 is ON, power will flow through this relay contact.

```
%I0122
 -| |-
```

For a coil, the reference represents a location in memory that is controlled by the flow of power into the coil. In this example, if power flows into the left side of the coil, reference %Q0004 is turned ON.

```
%Q0004
 -( )-
```

The programming software and the Hand-Held Programmer both have a coil check function that checks for multiple uses of %Q or %M references with relay coils or outputs on functions.

Format of Program Function Blocks (Instructions)

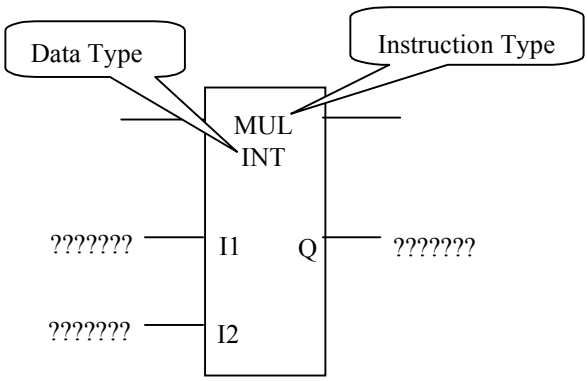
Some functions are very simple, like the Master Control Relay (MCR) function, which is shown with the abbreviated name of the function within brackets:

```
-[ MCR ]-
```

Other functions are more complex. They may have several places where you will enter information (parameter data) to be used by the function.

The example function block illustrated below is a multiplication (MUL) instruction. Its parts are typical of many function blocks. However, the number and types of parameters used can vary widely among the various type of function blocks. The upper part of the function block shows the name of the function. It may also show a data type, in this case, signed integer.

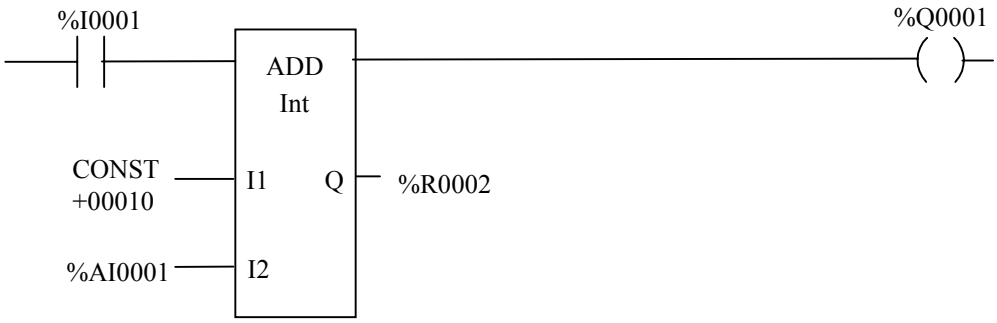
Many program functions (instructions) allow you to select the data type for the function after selecting the function. For example, the data type for the MUL function could be changed to double precision signed integer (D_INT). Additional information on data types is provided earlier in this chapter.



Function Block (Instruction) Parameters

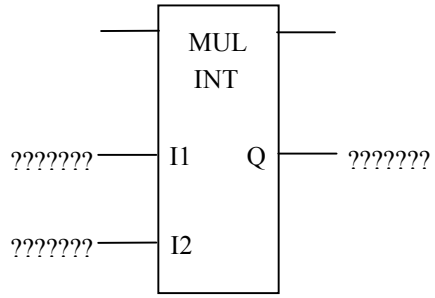
Each line entering the left side of a function block represents an input for that function. There are two forms of input used with function blocks, discrete and analog. Discrete inputs are either ON or OFF. In the figure below, the enabling contact %I0001 is an example of a discrete input. Analog inputs can be either constants or references. A constant is an explicit value. A reference is the memory address of a value. Generally, a reference is used if the input data is subject to change. For example, a reference might be the address of an input from an analog measuring device.

In the following example, input parameter I1 for an ADD function block is a constant, and input parameter I2 is a reference.



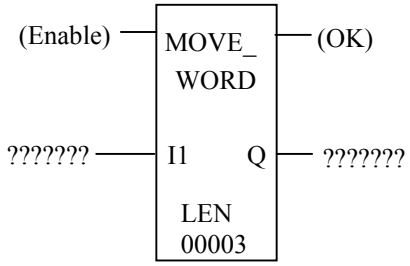
Each line exiting the right side of the function block represents an output. Outputs can be either discrete or analog. If analog, the value is placed into a register (reference). In the example above, the function block's OK output is discrete and it controls coil %Q0001. Its Q output, however, holds the resulting value from the math operation , so it is placed into a register, %R0002 in this example.

Where the question marks appear on the left of a function block, you will enter either the data itself, a reference location where the data is found, or a variable representing the reference location where the data is found. Where question marks appear on the right of a function block, you will usually enter a reference location for data to be output by the function block or a variable that represents the reference location for data to be output by the function block.

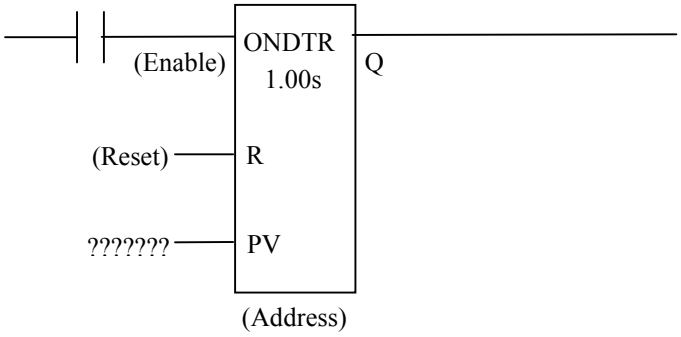


Most function blocks do not change input data; instead, they utilize input data in an operation and place the result of the operation in an output reference.

For functions that operate on groups of memory addresses (references), a length can be selected for the function. In the following function block, the LEN operand specifies the number of input words to be moved (3 in this example).

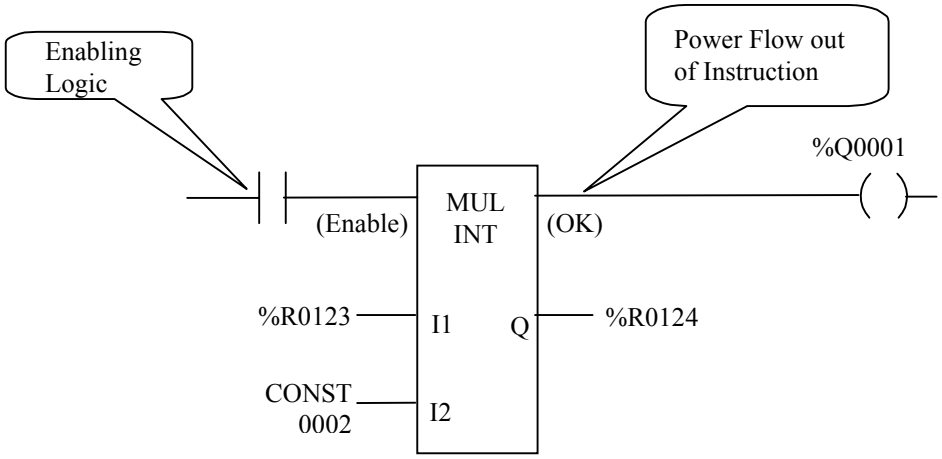


Timer, counter, BITSEQ, and ID functions require an address for the location of three words (registers) that store the current value, preset value, and a control word or "Instance" of the function. The first word of the three consecutive words appears on-screen below the function block, shown in the following figure as "(Address)."



Power Flow In and Out of a Function

Power flows into a function block's Enable input on the upper left through enabling logic. Most function blocks have a power flow output, called the "OK" output. If the function block executes properly, the OK output goes high and passes power flow out. If another device is connected to the OK output, such as the output coil shown below, that device is enabled. However, use of the OK output is optional for many function blocks, since their primary purpose is to obtain the result of the operation (multiplication in the example below) at the Q output.



Note

If using Logicmaster programming software, function blocks cannot be tied directly to the left power rail. You can use %S7, the ALW_ON (always on) bit with a normally open contact tied to the power rail to call a function every sweep.

Power flows out of the function block on the upper right. It may be passed to other program logic or to a coil (optional). Function blocks pass power when they execute successfully.

Section 3: Power-Up and Power-Down Sequences

There are two possible power-up sequences in the Series 90-30 PLC; a cold power-up and a warm power-up. The CPU normally uses the cold power-up sequence. However, in a Model 331 or higher PLC system, if the time that elapses between a power-down and the next power-up is less than five seconds, the warm power-up sequence is used.

Power-Up

A cold power-up consists of the following sequence of events. A warm power-up sequence skips Step 1.

1. The CPU will run self-diagnostics. This includes checking a portion of battery-backed RAM to determine whether or not the RAM contains valid data.
2. If an EPROM, EEPROM, or flash is present and the PROM power-up option in the PROM specifies that the PROM contents should be used, the contents of PROM are copied into RAM memory. If an EPROM, EEPROM, or flash is not present, RAM memory remains the same and is not overwritten with the contents of PROM.
3. The CPU interrogates each slot in the system to determine which boards are present.
4. The hardware configuration is compared with software configuration to ensure that they are the same. Any mismatches detected are considered faults and are alarmed. For example, if a module is specified in the software configuration but a different module is present in the actual hardware configuration, this condition is a fault and is alarmed.
5. If there is no software configuration, the CPU will use its built-in default configuration.
6. The CPU establishes the communications channel between itself and any intelligent modules.
7. In the final step of the execution, the mode of the first sweep is determined based on CPU configuration. Figure 2-5 on the next page shows the decision sequence for the CPU when it decides whether to copy from PROM or to power-up in **`STOP`** or **`RUN`** mode.

Note

Steps 2 through 7 above do not apply to the Series 90 Micro PLC. For information about the power-up and power-down sequences for the Micro, refer to the "Power-up and Power-down Sequences" section of Chapter 5, "System Operation," in the *Series 90 Micro PLC User's Manual* (GFK-1065).

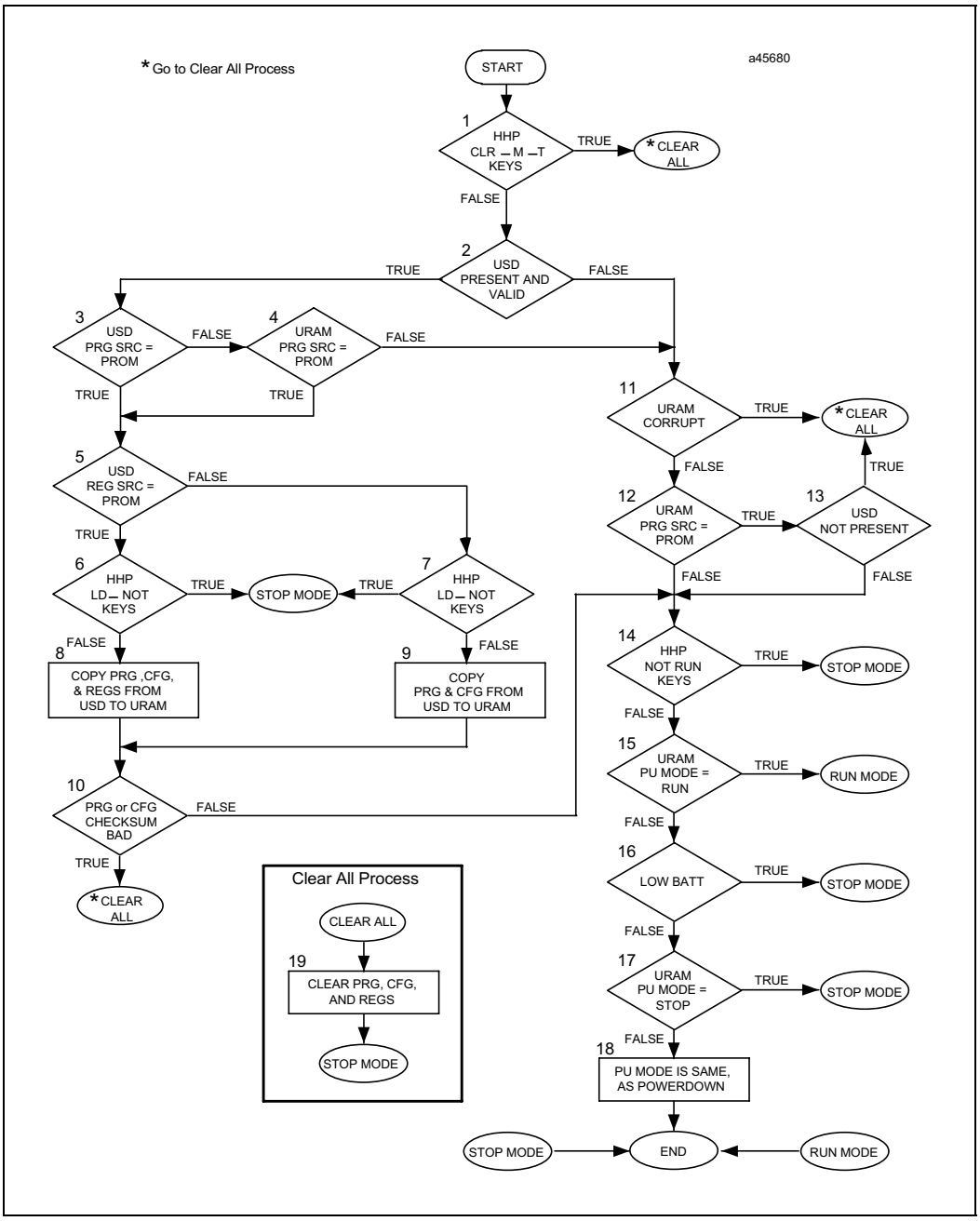


Figure 2-5. Power-Up Sequence

Prior to the START statement on the Power Up Flowchart, the CPU goes through power up diagnostics which test various peripheral devices used by the CPU and tests RAM. After completing diagnostics, internal data structures and peripheral devices used by the CPU get initialized. The CPU then determines if User Ram has been corrupted. If User Ram is corrupted, the user program and configuration are cleared out and defaulted and all user registers are cleared.

FLOW CHART TERMS:

PRG = User program (PRG SRC = Program Source)
CFG = User configuration
REGS = User registers (%I, %Q, %M, %G, %R, %AI, and %AQ references).
USD = User storage device, either an EPROM, EEPROM, or Flash device.
URAM = Non-volatile user ram which contains PRG, CFG, and REGS.
HHP = Hand-Help Programmer
PU = Power-up
CLR = Clear
BATT = Battery

FLOW CHART EXPANDED TEXT:

(1) Are the <CLR> and <M_T> keys being pressed on the HHP (Hand-Held Programmer) during power-up to clear all URAM?

(2) Is the USD (user storage device) present and is the information in the USD valid?

(3) Is the PRG SRC parameter in the USD set to Prom meaning to load the PRG (program logic) and CFG (configuration) from the USD device?

(4) Is the PRG SRC parameter in the URAM set to Prom meaning to load the PRG and CFG from the USD device?

(5) Is the REG SRC parameter in the USD set to Prom meaning to load the REGS (registers) from the USD device?

(6 & 7) Are the <LD> and <NOT> keys being pressed on the HHP during power-up to keep the PRG, CFG, and REGS from being loaded from USD?

(8) Copy PRG, CFG, and REGS from the USD to URAM.

(9) Copy PRG and CFG from the USD to URAM.

(10) Is the PRG or CFG checksums just loaded from USD invalid?

(11) Is the URAM corrupted? Could be due to being powered down without a battery attached or a low battery. Could also be due to updating firmware.

(12) Is the PRG SRC parameter in the URAM set to Prom meaning to load the PRG and CFG from the USD device?

(13) Is the USD present? This check only applies to CPUs 311-341. The USD is assumed to be present for CPUs 350-364 and 374.

(14) Are the <**NOT**> and <**RUN**> keys being pressed on the HHP during power-up to unconditionally power-up in Stop Mode?

(15) Is the PWR UP parameter in URAM set to **RUN**?

(16) Is the battery low?

(17) Is the PWR UP parameter in URAM set to **STOP**?

(18) Set the power up mode to what ever the power down mode was.

(19) Clear PRG, CFG, and REGS.

Note

The first part of this chart on the previous page does not apply to the Series 90 Micro PLC. For information about the power-up and power-down sequences for the Micro, refer to the "Power-up and Power-down Sequences" section of Chapter 5, "System Operation," in the *Series 90 Micro PLC User's Manual* (GFK-1065).

Power-Down

System power-down occurs when the power supply detects that incoming AC power has dropped for more than one power cycle or the output of the 5-volt power supply has fallen to less than 4.9 volts DC.

Section 4: Clocks and Timers

Clocks and timers provided by the Series 90-30 PLC include an elapsed time clock, a time-of-day clock (Models 331, 340/341, 350-374, and the 28-point Micro), a watchdog timer, and a constant sweep timer. Three types of timer function blocks include an on-delay timer, an off-delay timer, and a retentive on-delay timer (also called a watch clock timer). Four system time-tick contacts cycle on and off for 0.01 second, 0.1 second, 1.0 second, and 1 minute intervals.

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 recorded. This seconds count rolls over 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 can not be reset from the user program or the programmer. However, the application program can read the current value of the elapsed time clock by using Service Request 16.

Time-of-Day Clock

The time of day in the 28-point Micro and Series 90-30 PLC Model 331 and higher is maintained by a hardware time-of-day clock. The time-of-day clock maintains seven time functions:

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

The time-of-day (TOD) clock is battery-backed and maintains its present state across a power failure. However, unless you initialize the clock, the values it contains are meaningless. The application program can read and set the time-of-day clock using Service Request #7.

The time-of-day clock can also be read and set from the CPU configuration software and with the Hand-Held Programmer (HHP). However, starting with CPU (350-364) firmware release 8.00, if the CPU *Mem. Protect* parameter is set to *Enabled*, the HHP cannot change the TOD clock if the CPU keylock switch is in the ON position. Note that keylock protection features only apply to CPUs 350—374 (other CPUs do not have a keylock switch).

The time-of-day clock is designed to handle month-to-month and year-to-year transitions. It automatically compensates for leap years until the year 2079.

Watchdog Timer

A watchdog timer in the Series 90-30 PLC is designed to catch catastrophic failure conditions that result in an unusually long sweep. The timer value for the watchdog timer is 200 milliseconds for CPUs 311-341, and 500 milliseconds for CPUs 350—374; this is a fixed value that cannot be changed. The watchdog timer always starts from zero at the beginning of each sweep.

For 331 and lower model 90-30 CPUs, if the watchdog timeout value is exceeded, the OK LED goes off; the CPU is placed in reset and completely shuts down; and outputs go to their default state. No communication of any form is possible, and all microprocessors on all boards are halted. To recover, power must be cycled on the rack containing the CPU. In the 90-20, Series 90 Micro and 340 and higher 90-30 CPUs, a watchdog timeout causes the CPU to reset, execute its powerup logic, generate a watchdog failure fault, and change its mode to **STOP**.

Elapsed Power Down Timer

The elapsed power down timer is used to determining how long the PLC was powered off. When the PLC is powered off, it resets to 0 and starts to time. When the PLC is powered on, timing stops and the value is retained. Service Request #29, described in chapter 12, can be used to read the value of this timer.

Note

This function is available only in the 331 or higher Series 90-30 CPUs.

Constant Sweep Timer

The constant sweep timer controls the length of a program sweep when the Series 90-30 PLC operates in **CONSTANT SWEEP TIME** mode. In this mode of operation, each sweep consumes the same amount of time. The value of the constant sweep timer is set by the programmer and can be any value from 5 to the value of the watchdog timer. Constant Sweep Time default is 100 milliseconds. Typically, for most application programs, the input scan, application program logic scan, and output scan do not require exactly the same amount of execution time in each sweep.

If the constant sweep timer expires before the completion of the sweep and the previous sweep was not an oversweep, the PLC places an oversweep alarm in the PLC fault table. At the beginning of the next sweep, the PLC sets the OV_SWP fault contact. The OV_SWP contact is reset when the PLC is not in **CONSTANT SWEEP TIME** mode or the time of the last sweep did not exceed the constant sweep timer.

Time-Tick Contacts

The Series 90 PLC provides four time-tick contacts with time durations of 0.01 second, 0.1 second, 1.0 second, and 1 minute. The state of these contacts only changes during the housekeeping portion of the PLC sweep. These contacts provide a pulse having an equal on and off time duration. The contacts are referenced as T_10MS (0.01 second), T_100MS (0.1 second), T_SEC (1.0 second), and T_MIN (1 minute).

The following timing diagram represents the on/off time duration of these contacts.

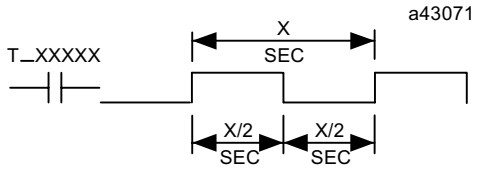


Figure 2-6. Time-Tick Contact Timing Diagram

Section 5: System Security

Security in Series 90-30, Series 90-20, and in the Micro PLCs is designed to prevent unauthorized changes to the contents of a PLC. There are four security levels available in the PLC. The first level, which is always available, provides only the ability to read PLC data; no changes are permitted to the application. The other three levels have access to each level protected by a password.

Each higher privilege level permits greater change capabilities than the lower level(s). Privilege levels accumulate in that the privileges granted at one level are a combination of that level, plus all lower levels. The levels and their privileges are:

Privilege Level	Description
Level 1	Any data, except passwords may be read. This includes all data memories (%I, %Q, %AQ, %R, etc.), fault tables, and all program block types (data, value, and constant). No values may be changed in the PLC.
Level 2	This level allows write access to the data memories (%I, %R, etc.).
Level 3	This level allows write access to the application program in **STOP** mode only.
Level 4	This is the default level for systems that have no passwords set. The default level for a system with passwords is to the highest unprotected level. This level, the highest, allows read and write access to all memories as well as passwords in both **RUN** and **STOP** mode. (Configuration data cannot be changed in **RUN** mode.)

Passwords

There is one password for each privilege level in the PLC. (No password can be set for level 1 access.) Each password may be unique; however, the same password can be used for more than one level. Passwords are one to four ASCII characters in length; they can only be entered or changed with the programming software or the Hand-Held Programmer.

A privilege level change is in effect only as long as communications between the PLC and the programmer are intact. There does not need to be any activity, but the communications link must not be broken. If there is no communication for 15 minutes, the privilege level returns to the highest unprotected level.

Upon connection to the PLC, the programming software requests the protection status of each privilege level from the PLC. The programming software then requests the PLC to move to the highest unprotected level, thereby giving the programming software access to the highest unprotected level without having to request any particular level. When the Hand-Held Programmer is connected to the PLC, the PLC reverts to the highest unprotected level.

Privilege Level Change Requests

A programmer requests a privilege level change by supplying the new privilege level and the password for that level. A privilege level change is denied if the password sent by the programmer does not agree with the password stored in the PLC's password access table for the requested level. The current privilege level is maintained and no change will occur. If you attempt to access or modify information in the PLC using the Hand-Held Programmer without the proper privilege level, the Hand-Held Programmer will respond with an error message that the access is denied.

Locking/Unlocking Subroutines

Subroutine blocks can be locked and unlocked using the block-locking feature of programming software. Two types of locks are available:

Type of Lock	Description
View	Once locked, you cannot zoom into that subroutine.
Edit	Once locked, the information in the subroutine cannot be edited.

A previously view locked or edit locked subroutine may be unlocked in the block declaration editor unless it is permanently view locked or permanently edit locked.

A search or search and replace function may be performed on a view locked subroutine. If the target of the search is found in a view locked subroutine, one of the following messages is displayed instead of logic:

```
Found in locked block <block_name>   (Continue/Quit)
```

or

```
Cannot write to locked block <block_name>  (Continue/Quit)
```

You may continue or abort the search.

Folders that contain locked subroutines may be cleared or deleted. If a folder contains locked subroutines, these blocks remain locked when the programming software Copy, Backup, and Restore folder functions are used.

Permanently Locking a Subroutine

In addition to VIEW LOCK and EDIT LOCK, there are two types of permanent locks. If a PERMANENT VIEW LOCK is set, all zooms into a subroutine are denied. If a PERMANENT EDIT LOCK is set, all attempts to edit the block are denied.

Caution

The permanent locks differ from the regular VIEW LOCK and EDIT LOCK in that once set, they cannot be removed.

Once a PERMANENT EDIT LOCK is set, it can only be changed to a PERMANENT VIEW LOCK. A PERMANENT VIEW LOCK cannot be changed to any other type of lock.

Section 6: Series 90-30, 90-20, and Micro I/O System

The PLC I/O system provides the interface between the Series 90-30 PLC and user-supplied devices and equipment. Series 90-30 I/O modules plug directly into slots in Series 90-30 baseplates. The number of Series I/O modules supported depends upon the CPU model:

- CPU models 350—374 support up to 79 I/O modules. These CPUs support up to eight racks, which includes the CPU rack plus a total of seven expansion and/or remote racks.
- CPU models 331, 340, and 341, support up to 49 I/O modules. These CPUs support up to five racks, which includes the CPU rack plus a total of four expansion and/or remote racks.
- CPU models 311 and 313 (5-slot baseplates) support up to 5 Series 90-30 I/O modules. CPU model 323 (10-slot baseplate) supports up to 10 Series 90-30 I/O modules. These three CPUs do not support expansion or remote racks.

The I/O structure for the Series 90-30 PLC is shown in the following figure.

PLC I/O System

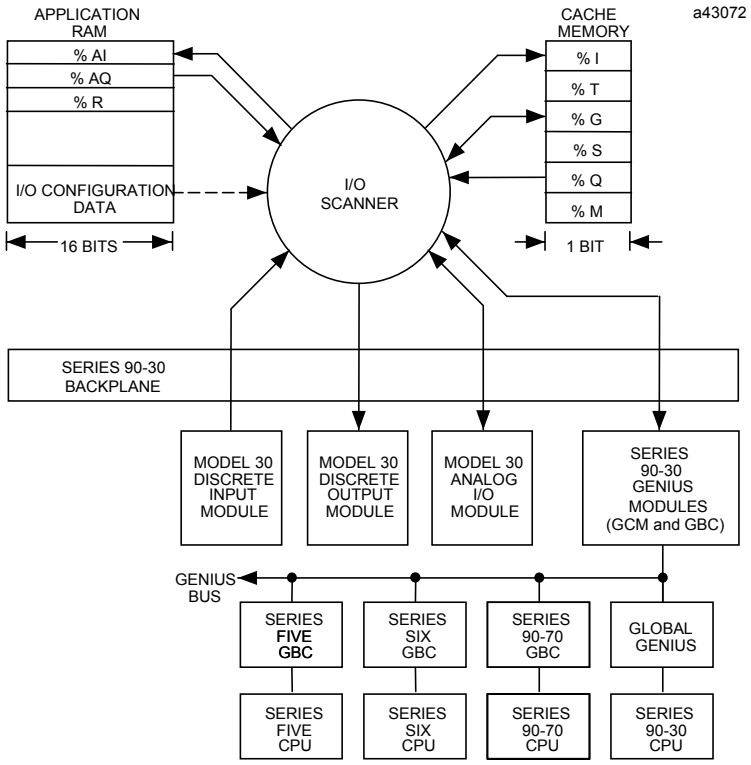


Figure 2-7. Series 90-30 I/O Structure

Note

The drawing shown above is specific to the 90-30 I/O structure. Intelligent and option modules are not part of the I/O scan; they use the System Communication Window. For information about the 90-20 I/O structure, refer to the *Series 90™-20 Programmable Controller User's Manual* (GFK-0551). For information about the Micro PLC I/O structure, refer to the *Series 90™ Micro PLC User's Manual* (GFK-1065).

Series 90-30 I/O Modules

Series 90-30 I/O modules are available as five types, discrete input, discrete output, analog input, analog output, and option modules. The following table lists the Series 90-30 I/O modules by catalog number, number of I/O points, and a brief description of each module.

Note

Contact your local GE Fanuc distributor for availability of the modules listed. Refer to the "**Pub Number"** column for publications that contain the specifications and wiring information of each Series 90-30 I/O module.

Figure 2-8. Series 90-30 I/O Modules

Catalog Number	Points	Description	Pub Number
		Discrete Modules - Input	
IC693MDL230	8	120 VAC Isolated	GFK-0898
IC693MDL231	8	240 VAC Isolated	GFK-0898
IC693MDL240	16	120 VAC	GFK-0898
IC693MDL241	16	24 VAC/DC Positive/Negative Logic	GFK-0898
IC693MDL630	8	24 VDC Positive Logic	GFK-0898
IC693MDL632	8	125 VDC Positive/Negative Logic	GFK-0898
IC693MDL633	8	24 VDC Negative Logic	GFK-0898
IC693MDL634	8	24 VDC Positive/Negative Logic	GFK-0898
IC693MDL640	16	24 VDC Positive Logic	GFK-0898
IC693MDL641	16	24 VDC Negative Logic	GFK-0898
IC693MDL643	16	24 VDC Positive Logic, FAST	GFK-0898
IC693MDL644	16	24 VDC Negative Logic, FAST	GFK-0898
IC693MDL645	16	24 VDC Positive/Negative Logic	GFK-0898
IC693MDL646	16	24 VDC Positive/Negative Logic, FAST	GFK-0898
IC693MDL652	32	24 VDC Position/Negative Logic	GFK-0898
IC693MDL653	32	24 VDC Positive/Negative Logic, FAST	GFK-0898
IC693MDL654	32	5/12 VDC (TTL) Positive/Negative Logic	GFK-0898
IC693MDL655	32	24 VDC Positive/Negative Logic	GFK-0898
IC693ACC300	8/16	Input Simulator	GFK-0898

Table 2-8. Series 90-30 I/O Modules - Continued

Catalog Number	Points	Description	Pub Number
		Discrete Modules - Output	
IC693MDL310	12	120 VAC, 0.5A	GFK-0898
IC693MDL330	8	120/240 VAC, 2A	GFK-0898
IC693MDL340	16	120 VAC, 0.5A	GFK-0898
IC693MDL390	5	120/240 VAC Isolated, 2A	GFK-0898
IC693MDL730	8	12/24 VDC Positive Logic, 2A	GFK-0898
IC693MDL731	8	12/24 VDC Negative Logic, 2A	GFK-0898
IC693MDL732	8	12/24 VDC Positive Logic, 0.5A	GFK-0898
IC693MDL733	8	12/24 VDC Negative Logic, 0.5A	GFK-0898
IC693MDL734	6	125 VDC Positive/Negative Logic, 2A	GFK-0898
IC693MDL740	16	12/24 VDC Positive Logic, 0.5A	GFK-0898
IC693MDL741	16	12/24 VDC Negative Logic, 0.5A	GFK-0898
IC693MDL742	16	12/24 VDC Positive Logic, 1A	GFK-0898
IC693MDL750	32	12/24 VDC Negative Logic	GFK-0898
IC693MDL751	32	12/24 VDC Positive Logic, 0.3A	GFK-0898
IC693MDL752	32	5/24 VDC (TTL) Negative Logic, 0.5A	GFK-0898
IC693MDL753	32	12/24 VDC Positive/Negative Logic, 0.5A	GFK-0898
IC693MDL760	16	11 Pneumatic and five 24VDC Positive Logic, 0.5 A	GFK-1881
IC693MDL930	8	Relay, N.O., 4A Isolated	GFK-0898
IC693MDL931	8	Relay, BC, Isolated	GFK-0898
IC693MDL940	16	Relay, N.O., 2A	GFK-0898
		Input/Output Modules	
IC693MDR390	8/8	24 VDC Input, Relay Output	GFK-0898
IC693MAR590	8/8	120 VAC Input, Relay Output	GFK-0898
		Analog Modules	
IC693ALG220	4 ch	Analog Input, Voltage	GFK-0898
IC693ALG221	4 ch	Analog Input, Current	GFK-0898
IC693ALG222	16	Analog Input, Voltage	GFK-0898
IC693ALG223	16	Analog Input, Current	GFK-0898
IC693ALG390	2 ch	Analog Output, Voltage	GFK-0898
IC693ALG391	2 ch	Analog Output, Current	GFK-0898
IC693ALG392	8 ch	Analog Output, Current/Voltage	GFK-0898
IC693ALG442	4/2	Analog, Current/Voltage Combination Input/Output	GFK-0898

Table 2-8. Series 90-30 I/O Modules - Continued

Catalog Number	Description	Pub Number
	Option Modules	
IC693APU300	High Speed Counter	GFK-0293
IC693APU301 IC693APU301	Motion Mate APM Module, 1-Axis–Follower Mode Motion Mate APM Module, 1-Axis–Standard Mode	GFK-0781 GFK-0840
IC693APU302 IC693APU302 IC693MCS001/002*	Motion Mate APM Module, 2-Axis–Follower Mode Motion Mate APM Module, 2-Axis–Standard Mode Power Mate J Motion Control System (1 and 2 Axis)	GFK-0781 GFK-0840 GFK-1256
IC693DSM302 IC693DSM314	Motion Mate Digital Servo Module Motion Mate Digital Servo Module	GFK-1464 GFK-1742
IC693APU305 IC693CMM321	I/O Processor Module Ethernet Communications Module	GFK-1028 GFK-1541
IC693ADC311	Alphanumeric Display Coprocessor	GFK-0521
IC693BEM331	Genius Bus Controller	GFK-1034
IC693BEM320	I/O Link Interface Module (slave)	GFK-0631
IC693BEM321	I/O Link Interface Module (master)	GFK-0823
IC693CMM311	Communications Coprocessor Module	GFK-0582
IC693CMM301	Genius Communications Module	GFK-0412
IC693CMM302	Enhanced Genius Communications Module	GFK-0695
IC693PBM200	Profibus Master Module	GFK-2121
IC693PBS201	Profibus Slave Module	GFK-2193
IC693PCM300	PCM, 160K Bytes (35Kbytes User MegaBasic Program)	GFK-0255
IC693PCM301	PCM, 192K Bytes (47Kbytes User MegaBasic Program)	GFK-0255
IC693PCM311	PCM, 640K Bytes (190Kbytes User MegaBasic Program)	GFK-0255
IC693PTM100/101	Power Transducer Module (PTM)	GFK-1734
IC693TCM302/303	Temperature Control Module (TCM), eight-channel	GFK-1466

* Obsolete. Listed for reference only.

I/O Data Formats

Discrete inputs and discrete outputs are stored as bits in bit cache (status table) memory. Analog input and analog output data are stored as words and are memory resident in a portion of application RAM memory allocated for that purpose.

Default Conditions for Series 90-30 Output Modules

At power-up, Series 90-30 discrete output modules default to outputs off. They will retain this default condition until the first output scan from the PLC. Analog output modules can be configured with a jumper located on the module's removable terminal block to either default to zero or retain their last state. Also, analog output modules may be powered from an external power source so that, even if the PLC has no power, the analog output modules will continue to operate in their selected default state.

Diagnostic Data

Diagnostic bits are available in %S memory that will indicate the loss of an I/O module or a mismatch in I/O configuration. Diagnostic information is not available for individual I/O points. More information on fault handling can be in Chapter 3, “Fault Explanations and Correction.”

Global Data

Genius Global Data

The Series 90-30 PLC supports very fast sharing of data between multiple CPUs using Genius global data. The Genius Bus Controller, IC693BEM331 in CPU, version 5 and later, and the Enhanced Genius Communications Module, IC693CMM302, can broadcast up to 128 bytes of data to other PLCs or computers. They can receive up to 128 bytes from each of the up to 30 other Genius controllers on the network. Data can be broadcast from or received into any memory type, not just %G global bits.

The original Genius Communications Module, IC693CMM301, is limited to fixed %G addresses and can only exchange 32 bits per serial bus address from SBA 16 to 23. For new installations, we recommend this module not be used; instead, use the newer enhanced GCM, which has considerably more capability.

Global data can be shared between Series Five, Series Six, and Series 90 PLCs connected to the same Genius I/O bus.

Ethernet Communications

The Model 364 CPU (release 9.0 and later) supports connection to an Ethernet network through either (but not both) of two built-in Ethernet ports. AAUI and 10BaseT ports are provided. The Model 374 CPU supports connection to an Ethernet network through two built-in 10BaseT/100BaseTx auto-negotiating full-duplex Ethernet ports.

Both the CPU364 and CPU374 support Ethernet Global Data (EGD), which is similar to Genius Global Data in that it allows one device (the producer) to transfer data to one or more other devices (the consumers) on the network. EGD is not supported by Logicmaster 90 software (requires a Windows-based programmer for Series 90 PLCs.)

Series 90-20 I/O Modules

The following I/O modules are available for the Series 90-20 PLC. Each module is listed by catalog number, number of I/O points, and a brief description. The I/O is integrated into a baseplate along with the power supply. For the specifications and wiring information of each module, refer to chapter 5 in the *Series 90-20 Programmable Controller User's Manua*l, GFK-0551.

Catalog Number	Description	I/O Points
IC692MAA541	I/O and Power Supply Base Module, 120 VAC In/120 VAC Out/120 VAC Power Supply	16 In/12 Out
IC692MDR541	I/O and Power Supply Base Module 24 VDC In/Relay Out/120 VAC Power Supply	16 In/12 Out
IC692MDR741	I/O and Power Supply Base Module 24V DC In/Relay Out/240 VAC Power Supply	16 In/12 Out
IC692CPU211	CPU Module, Model CPU 211	Not Applicable

Configuration and Programming

Configuration is the process of assigning logical addresses, as well as other characteristics, to the hardware modules in the system. It can be done either before or after programming, using the configuration software or Hand-Held Programmer; however, it is recommended that configuration be done first. Refer to the User's Manual for your programming software for details on how to create, transfer, edit, and print programs. Chapters 4 through 12 describe the programming instructions that can be used to create ladder logic programs for the Series 90-30 and Series 90-20 programmable controllers.

.

Chapter 3 Fault Explanation and Correction

This chapter is an aid to troubleshooting the Series 90-30, 90-20, and Micro PLC systems. 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 1: Fault Handling

Note

This information on fault handling applies to systems programmed using Logicmaster 90-30/20/Micro software.

Faults occur in the Series 90-30, 90-20, or Series 90 Micro PLC system when certain failures or conditions happen that 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. Or, a reported condition may only act as an alert, such as a low battery signal, which indicates that the memory backup battery needs to be changed. However, some conditions reported in the fault tables are not reports of failures. For example, if you were to add a new module to the PLC, this would be listed in the I/O fault table as "Addition of I/O Module."

Alarm Processor

A **fault** is the condition or failure itself. When a fault is received and processed by the CPU, it is called an **alarm**. The firmware in the CPU that handles these conditions is called the Alarm Processor. The user interface 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 90-30, 90-20, and Micro PLCs detect several classes of faults. These include internal failures, external failures, and operational failures.

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.
Operational Failures	Communication failures. Configuration failures. Password access failures.

Note

For information specific to Micro PLC fault handling, refer to the Series *90 Micro PLC User's Manual* (GFK-1065).

System Reaction to Faults

Hardware failures require that either the system be shut down or the failure be tolerated. I/O failures may be tolerated by the PLC system, but they may be intolerable by the application or the process being controlled. Operational failures are normally tolerated. Series 90-30, 90-20, and Micro PLC faults have two attributes:

Attribute	Description
Fault Table Affected	I/O Fault Table PLC Fault Table
Fault Action	Fatal Diagnostic Informational

Fault Tables

Two fault tables are maintained in the PLC for logging faults, the I/O fault table for logging faults related to the I/O system and the PLC fault table for logging all other faults. The following table lists the fault groups, their fault actions, the fault tables affected, and the "name" for system discrete %S points that are affected.

Table 3-1. Fault Summary

Fault Group	Fault Action	Fault Table	Special Discrete Fault References			
Loss of or Missing I/O Module	Diagnostic	I/O	io_flt	any_flt	io_pres	los_iom
Loss of or Missing Option Module	Diagnostic	PLC	sy_flt	any_flt	sy_pres	los_sio
System Configuration Mismatch	Fatal	PLC	sy_flt	any_flt	sy_pres	cfg_mm
PLC CPU Hardware Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	hrd_cpu
Program Checksum Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	pb_sum
Low Battery	Diagnostic	PLC	sy_flt	any_flt	sy_pres	low_bat
PLC Fault Table Full	Diagnostic	—	sy_full			
I/O Fault Table Full	Diagnostic	—	io_full			
Application Fault	Diagnostic	PLC	sy_flt	any_flt	sy_pres	apl_flt
No User Program	Informational	PLC	sy_flt	any_flt	sy_pres	no_prog
Corrupted User RAM	Fatal	PLC	sy_flt	any_flt	sy_pres	bad_ram
Password Access Failure	Diagnostic	PLC	sy_flt	any_flt	sy_pres	bad_pwd
PLC Software Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	sft_cpu
PLC Store Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	stor_er
Constant Sweep Time Exceeded	Diagnostic	PLC	sy_flt	any_flt	sy_pres	ov_swp
Unknown PLC Fault	Fatal	PLC	sy_flt	any_flt	sy_pres	
Unknown I/O Fault	Fatal	I/O	io_flt	any_flt	io_pres	

Fault Action

Faults can be fatal, diagnostic or informational.

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

Possible fault actions are listed in the following table.

Table 3-2. Fault Actions

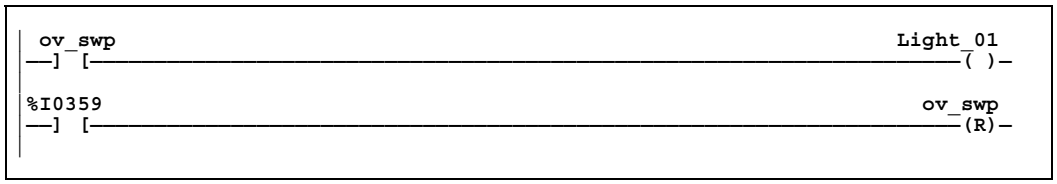
Fault Action	Response by CPU
Fatal	Log fault in fault table. Set fault references. Go to **STOP** mode.
Diagnostic	Log fault in fault table. Set fault references.
Informational	Log fault in fault table.

When a fault is detected, the CPU uses the fault action for that fault. Fault actions are not configurable in the Series 90-30 PLC, Series 90-20, or the Series 90 Micro PLC.

Fault References

System fault references in the Series 90-30 are of one type - fault summary references. Fault summary references are set to indicate what fault occurred. The fault reference remains on until the PLC is cleared or until cleared by the application program.

An example of a system fault bit being set and then cleared is shown in the following figure. In this example, the coil, Light_01, is turned on when system contact OV_SWP (%SA0002) closes, which indicates that an oversweep occurred. The OV_SWP contact and Light_01 coil are turned off if contact %I0359 is closed, because closing %I0359 turns on reset coil OV_SWP.



System Status References

The alarm processor maintains the states of the 128 system status bits in %S memory. Many of these status references indicate where a fault has occurred and what type of fault it is. Status references are assigned to %S, %SA, %SB, and %SC memory, and each reference has a nickname. For example, status bit %SA0009 has a nickname of CFG_MM, and it goes high to indicate a

configuration mismatch. These references are available for use in the application program as required. Refer to Chapter 2, "System Operation," for a list of the system status references.

Additional Fault Effects

Two faults described later in this chapter have additional effects associated with them. These effects are discussed in the following table.

Fault	Effect Description
PLC CPU Software Failure	When a PLC CPU software failure is logged, the Series 90-30 or 90-20 CPU immediately transitions into a special **ERROR SWEEP** mode. No activity is permitted in this mode. The only method of clearing this condition is to reset the PLC by cycling power.
PLC Sequence Store Failure	If, while performing a store to the PLC, communication between the PLC and the programmer is interrupted or any other failure occurs which terminates the download (store), the PLC Sequence Store Failure fault is logged. As long as this fault is present in the system, the PLC will not transition to **RUN** mode. To resume operation, the error must be cleared. This can be accomplished by clearing the fault on the applicable Fault Table Screen of the programming software.

PLC Fault Table Display

The PLC Fault Table screen displays PLC faults such as password violations, PLC/configuration mismatches, parity errors, and communications errors.

Faults are stored in the PLC, so if the programming software is in the **OFFLINE** mode, no faults are displayed in this fault table. If the programming software is in either the **ONLINE** or **MONITOR** mode, PLC fault data is displayed. In **ONLINE** mode, faults can be cleared, although this feature may be password protected.

Once cleared, faults that are still present are not logged again in the table (except for the "Low Battery" fault) unless power is cycled or a new configuration is stored.

I/O Fault Table Display

The I/O Fault Table screen displays I/O faults such as circuit faults, address conflicts, forced circuits, and I/O bus faults.

Faults are stored in the PLC, so if the programming software is in the **OFFLINE** mode, no faults are displayed in this fault table. If the programming software is in either the **ONLINE** or **MONITOR** mode, I/O fault data is displayed. In **ONLINE** mode, faults can be cleared, although this feature may be password protected.

Once cleared, faults that are still present are not logged again in the table unless power is cycled or a new configuration is stored.

Accessing Additional Fault Information

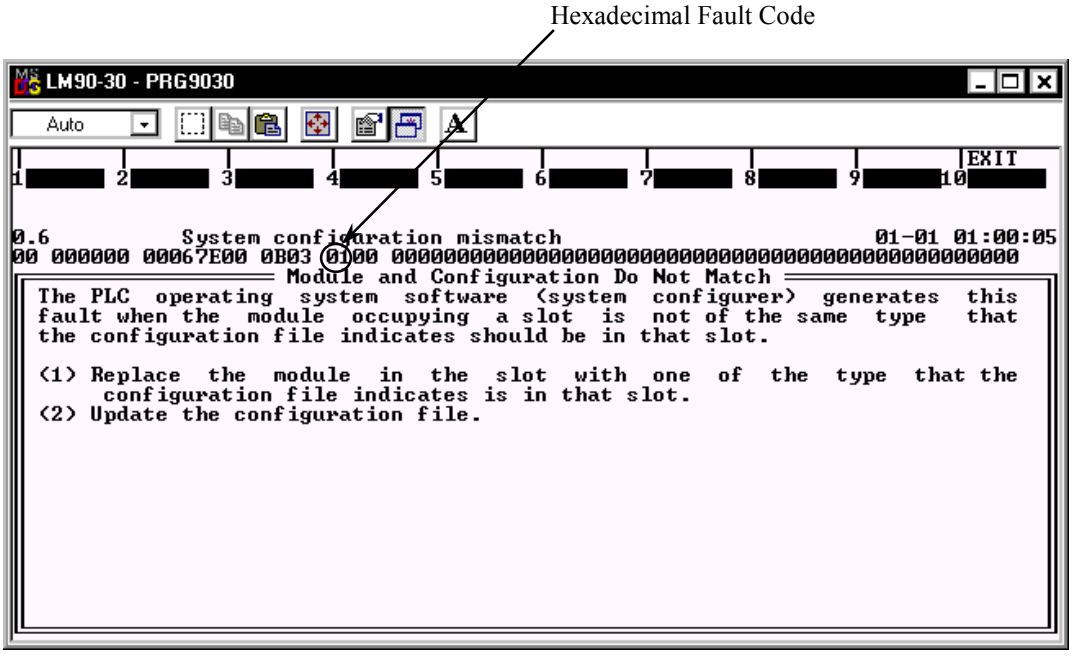
The fault tables contain basic information regarding the fault. Additional information pertaining to each fault can be displayed through the programming software. In addition, the programming software provides a hexadecimal fault code for each fault.

The last item, "Correction", in 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 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 fault code. Field Service personnel will then give you further instructions for the appropriate action to be taken.

The following figure of a Logicmaster fault detail screen shows the additional fault information and hexadecimal fault code discussed above. (The fault code is the first two hexadecimal digits in the fifth group of number from the left.) To reach this screen, select a Fault Table fault (Loss of I/O Module) by using the keyboard cursor control arrow keys, and then "zoom" using the F10 key. To return to the Fault Table screen, press either the Escape key or the Shift and F10 key combination.



Section 2: 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, displayed with the additional fault information, is used to distinguish different fault conditions sharing the same fault description. The error code is the first two hexadecimal digits in the fifth group (from the left) 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 the PLC CPU 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 PLC CPU board. The integrated circuits on the PLC CPU board may be failing.

The following table enables you to quickly find a particular PLC fault explanation in this section. Each entry is listed as it appears on the programmer screen.

Fault Actions

- **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; the PLC stays in ***RUN*** mode.
- **Informational** faults are simply logged in the PLC fault table; the PLC stays in ***RUN*** mode.

Loss of, or Missing, Option Module

The Fault Group **Loss of, or Missing Option Module** occurs when an option module 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 fault action for this group is **Diagnostic**.

Error Code:	1, 42
Name:	Option Module Soft Reset Failed
Description:	PLC CPU unable to re-establish communications with option module after soft reset (such as pressing a Reset button) is tried.
Correction:	(1) Repeat soft reset procedure recommended for this module. (2) Replace the option module. (3) Power off the system. Verify that the module is seated properly in the rack and that all cables are properly connected and seated. (4) Test or replace the cables.
Error Code:	All Others
Name:	Module Failure During Configuration
Description:	The PLC operating software 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.

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, ADC, etc.) comes online, is reset, or a module is found in the rack, but none is specified in the configuration. The fault action for this group is **Diagnostic**.

Correction:	(1) Update the configuration file to include the module. (2) Remove the module from the system.

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 fault action is **Fatal**.

Error Code:	1
Name:	System Configuration Mismatch
Description:	The PLC operating software generates this fault when the module occupying a slot is not of the same type that the configuration file indicates should be in that slot, or when the configured rack type does not match the actual rack present.
Correction:	Identify the mismatch and reconfigure the module or rack.
Error Code:	6
Name:	System Configuration Mismatch
Description:	This is the same as error code 1 in that this fault occurs when the module occupying a slot is not of the same type that the configuration file indicates should be in that slot, or when the configured rack type does not match the actual rack present.
Correction:	Identify the mismatch and reconfigure the module or rack.
Error Code:	18
Name:	Unsupported Hardware
Description:	A PCM or PCM-type module is present in a CPU 311, 313, or 323 system, or in an expansion or remote rack.
Correction:	Physically correct the situation by removing the PCM or PCM-type module or install a CPU that does support the module. NOTE: These modules must reside only in a CPU rack and only with a CPU that supports them.
Error Code:	26
Name:	Module busy–config not yet accept by module
Description:	The module cannot accept new configuration at this time because it is busy with a different process.
Correction:	Allow the module to complete the current operation and re-store the configuration.
Error Code:	51
Name:	END Function Executed from Sequential Function Chart (SFC) Action
Description:	The placement of an END function in SFC logic or in logic called by SFC will produce this fault.
Correction:	Remove the END function from the SFC logic or logic being called by the SFC logic.

Option Module Software Failure

The Fault Group **Option Module Software Failure** occurs when a non-recoverable software failure occurs on a PCM or ADC module. The fault action for this group is **Fatal**.

Error Code:	All
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.

Program Block Checksum Failure

The Fault Group **Program Block Checksum Failure** occurs when the PLC CPU detects error conditions in program blocks received by the PLC (downloaded by the programming software). It also occurs when the PLC CPU detects checksum errors during power-up verification of memory or during **RUN** mode background checking. The fault action for this group is **Fatal**.

Error Code:	All
Name:	Program Block Checksum Failure
Description:	The PLC Operating Software generates this error when a program block is corrupted.
Correction:	(1) Clear PLC memory and retry the store. (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.

Low Battery Signal

The Fault Group **Low Battery Signal** occurs when the PLC CPU detects a low battery on the PLC power supply or a module, such as the PCM, reports a low battery condition. The 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.
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.

Constant Sweep Time Exceeded

The Fault Group **Constant Sweep Time Exceeded** occurs when the PLC CPU operates in **CONSTANT SWEEP** mode, and it detects that the sweep has exceeded the constant sweep timer. The fault extra data contains the actual time of the sweep in the first two bytes and the name of the program in the next eight bytes. The fault action for this group is **Diagnostic**.

Correction:	(1)	Increase constant sweep time.
	(2)	Remove logic from application program.

Application Fault

The Fault Group **Application Fault** occurs when the PLC CPU detects a fault in the user program. The fault action for this group is **Diagnostic**, except when the error is a Subroutine Call Stack Exceeded, in which case it is **Fatal**.

Error Code:	7
Name:	Subroutine Call Stack Exceeded
Description:	Subroutine calls are limited to a depth of 8. A subroutine can call another subroutine which, in turn, can call another subroutine until 8 call levels are attained.
Correction:	Modify program so that subroutine call depth does not exceed 8.
Error Code:	1B
Name:	CommReq Not Processed Due To PLC Memory Limitations
Description:	No-wait communication requests can be placed in the queue faster than they can be processed (e.g., one per sweep). In a situation like this, when the communication requests build up to the point that the PLC has less than a minimum amount of memory available, the communication request will be faulted and not processed
Correction:	Issue fewer communication requests or otherwise reduce the amount of mail being exchanged within the system.
Error Code:	5A
Name:	User Shut Down Requested
Description:	The PLC operating software (function blocks) generates this informational alarm when Service Request #13 (User Shut Down) executes in the application program.
Correction:	None required. Information-only alarm.

No User Program Present

The Fault Group **No User Program Present** occurs when the PLC CPU is instructed to transition from **STOP** to **RUN** mode or a store to the PLC and no user program exists in the PLC. The PLC CPU detects the absence of a user program on power-up. The 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 fault action for this group is **Fatal**.

Error Code:	1
Name:	Corrupted User RAM on Power-Up
Description:	The PLC operating software (operating software) generates this error when it detects corrupted user RAM on power-up.
Correction:	(1) Reload the configuration file, user program, and references (if any). (2) Replace the battery on the PLC CPU. (3) Replace the expansion memory board on the PLC CPU. (4) Replace the PLC CPU.
Error Code:	2
Name:	Illegal Boolean OpCode Detected
Description:	The PLC operating software (operating software) generates this error when it detects a bad instruction in the user program.
Correction:	(1) Restore the user program and references (if any). (2) Replace the expansion memory board on the PLC CPU. (3) Replace the PLC CPU.

Password Access Failure

The Fault Group **Password Access 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 fault action for this group is **Informational**.

Correction:	Retry the request with the correct password.

PLC CPU System Software Failure

Faults in the Fault Group **PLC CPU System Software Failure** are generated by the operating firmware of the Series 90-30, 90-20 or Micro PLC CPU. They can occur at many different points of system operation. When a **Fatal** fault occurs, the PLC CPU **immediately** transitions into a special **ERROR SWEEP** mode. No activity is permitted when the PLC is in this mode. The only way to clear this condition is to cycle power on the PLC. The fault action for this group is **Fatal**.

Error Code:	1 through B
Name:	User Memory Could Not Be Allocated
Description:	The PLC operating software (memory manager) generates these errors when software requests the memory manager to allocate or de-allocate a block or blocks of memory from user RAM that are not legal. These errors should not occur in released products; they are normally encountered only during the firmware development process at the factory.
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.
Error Code:	D
Name:	System Memory Unavailable
Description:	The PLC operating software (I/O Scanner) generates this error when its request for a block of system memory is denied by the memory manager because no memory is available from the system memory heap. It is *Informational* if the error occurs during the execution of a DO I/O function block. It is *Fatal* if it occurs during power-up initialization or autoconfiguration.
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.
Error Code:	E
Name:	System Memory Could Not Be Freed
Description:	The PLC operating software (I/O Scanner) generates this error when it requests the memory manager to de-allocate a block of system memory and the de-allocation fails. This error can only occur during the execution of a DO I/O function block.
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:	10
Name:	Invalid Scan Request of the I/O Scanner
Description:	The PLC operating software (I/O Scanner) generates this error when the operating system or DO I/O function block scan requests neither a full nor a partial scan of the I/O. This 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.
Error Code:	13
Name:	PLC Operating Software Error
Description:	The PLC operating software generates this error when certain PLC operating software problems occur. This error should not occur in released products; they are normally encountered only during the firmware development process at the factory.
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:	14, 27
Name:	Corrupted PLC Program Memory
Description:	The PLC operating software generates these errors when certain PLC operating software problems occur. These errors should not occur in released products; they are normally encountered only during the firmware development process at the factory.
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:	27 through 4E
Name:	PLC Operating Software Error
Description:	The PLC operating software generates these errors when certain PLC operating software problems occur. These errors should not occur in released products; they are normally encountered only during the firmware development process at the factory.
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.
Error Code:	4F
Name:	Communications Failed
Description:	The PLC operating software (service request processor) generates this error when it attempts to comply with a request that requires backplane communications and receives a rejected response.
Correction:	(1) Check the bus for abnormal activity. (2) Replace the intelligent option module to which the request was directed.
Error Code:	50, 51, 53
Name:	System Memory Errors
Description:	The PLC operating software generates these errors when its request for a block of system memory is denied by the memory manager because no memory is available or contains errors.
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 operating software (service request processor) 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:	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.

Communications Failure During Store

The Fault Group **Communications Failure During Store** occurs during the store of program blocks and other data to the PLC. If communications with the programming device performing the store is interrupted or any other failure occurs which terminates the load, 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 order the condition to be cleared. The fault action for this group is **Fatal**. For additional information on this fault, please see the "Additional Fault Effects" section earlier in this chapter.

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

Section 3: I/O Fault Table Explanations

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

- Fault category.
- Fault type.
- Fault description.

The faults described on the following page have a fault category, but do not have a fault type or fault group.

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, displayed with the additional fault information obtained by pressing CTRL-F, is used to distinguish different fault conditions sharing the same fault description. (For more information about using CTRL-F, refer to Appendix B, "Interpreting Fault Tables," in this manual.) The Fault Category is the first two hexadecimal digits in the fifth group of numbers, as shown in the following example.

```
02  1F0100  00030101FF7F  0302  0200   84000000000003
                                 |
                                 |_____  Fault Category (first two hex
                                            digits in fifth group)
```

The following table enables you to quickly find a particular I/O fault explanation in this section. Each entry is listed as it appears on the programmer screen.

Loss of I/O Module

The Fault Category **Loss of I/O Module** applies to Model 30 discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category. The fault action is **Diagnostic**.

Description:	The PLC operating software generates this error when it detects that a Model 30 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 30 discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category. The fault action is **Diagnostic**.

Description:	The PLC operating software generates this error when an I/O module which had been faulted returns to operation.
Correction:	(1) No action necessary if the module was removed or replaced, or the remote rack was power cycled. (2) Update the configuration file or remove the module.
Description:	The PLC operating software generates this error when it detects a Model 30 I/O module in a slot which the configuration file indicates should be empty.
Correction:	(1) Remove the module if it is there by mistake. (2) Update and restore the configuration file to include the extra module if it is supposed to be there.

Chapter 4 Relay Functions

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.

Table 4-1. Types of Contacts

Type of Contact	Display	Contact Passes Power to Right
Normally Open	—| |—	When reference is ON.
Normally Closed	—|/|—	When reference is OFF.
Continuation Contact	<+>———	If the preceding continuation coil is set ON.

Using Coils

Coils are used to control discrete references such as %Q and %M memory types. 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 (contact) 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
Normally Open	—()—	ON	Sets reference ON.
		OFF	Sets reference OFF.
Negated	—(/)—	ON	Sets reference OFF.
		OFF	Sets reference ON.
Retentive	—(M)—	ON	Sets reference ON, retentive.
		OFF	Sets reference OFF, retentive.
Negated Retentive	—(/M)—	ON	Sets reference OFF, retentive.
		OFF	Sets reference ON, retentive.
Positive Transition	—(↑)—	OFF→ON	If reference is OFF, sets it ON for one sweep.
Negative Transition	—(↓)—	ON←OFF	If reference is OFF, sets it ON for one sweep.
SET	—(S)—	ON	Sets reference ON until reset OFF by —(R)—.
		OFF	Does not change the coil state.
RESET	—(R)—	ON	Sets reference OFF until set ON by —(S)—.
		OFF	Does not change the coil state.
Retentive SET	—(SM)—	ON	Sets reference ON until reset OFF by —(RM)—, retentive.
		OFF	Does not change the coil state.
Retentive RESET	—(RM)—	ON	Sets reference OFF until set ON by —(SM)—, retentive.
		OFF	Does not change the coil state.
Continuation Coil	——<+>	ON	Sets next continuation contact ON.
		OFF	Sets next continuation contact OFF.

Normally Open Contact —| |—

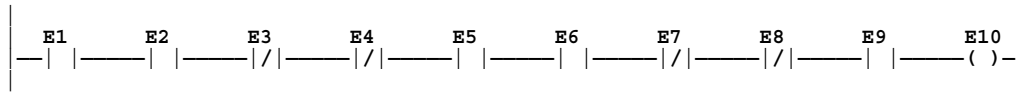
A normally open contact acts as a switch that passes power flow if the associated reference is ON (at logic 1).

Normally Closed Contact —|/|—

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

Example

The following example shows a rung with 10 elements having nicknames (see Chapter 2 for information on 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.

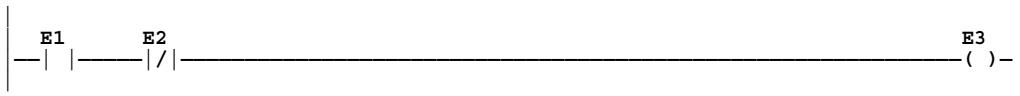


Coil —( )—

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 global Genius references (%G).

Example

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

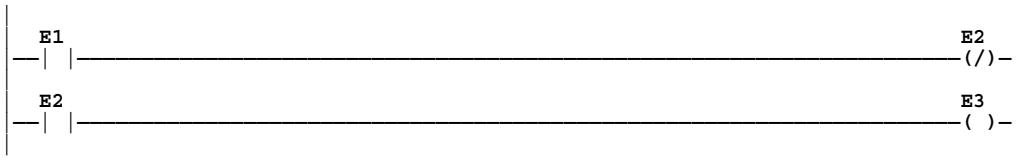


Negated Coil —(/)—

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 global Genius references (%G).

Example

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



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. Any contacts associated with that coil will change state for one PLC scan (sweep). (If the rung containing the coil is skipped on subsequent sweeps, it will remain ON.) This coil can be used as a one-shot.

Each reference should only be used as a transition coil once in the application program, so as to preserve the one-shot nature of 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 —(↓)—

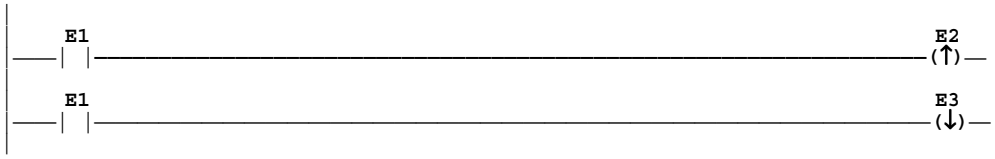
If the reference associated with this coil is OFF, when the coil ***stops*** receiving power flow, the reference is set to ON and any contacts associated with that coil will change state for one sweep.

A reference used with a transition coil should only be used as a coil once in the application program, so as to preserve the one-shot nature of 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.



SET Coil —(S)—

SET and RESET are non-retentive coils that can be used to keep ("latch") the state of a reference either ON or OFF. When a SET coil receives power flow, its reference stays ON (whether or not the coil itself continues to receive power flow) until the reference is reset by another coil.

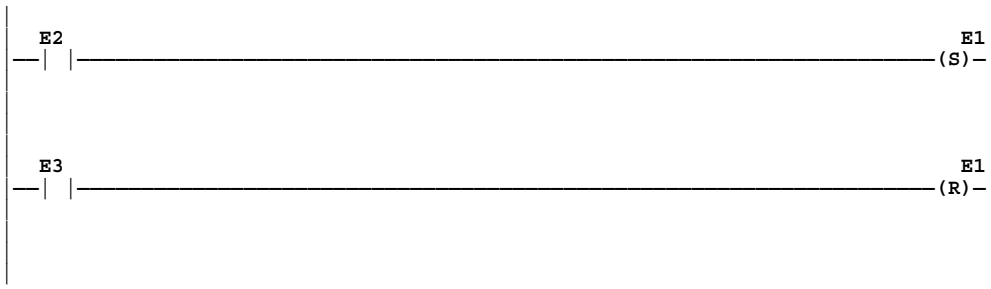
RESET Coil —(R)—

The RESET coil sets a discrete reference OFF if the coil receives power flow. The reference remains OFF until the reference is reset by another coil. The last-solved SET coil or RESET coil of a pair takes precedence.

Example

In the following example, the coil represented by E1(S) is turned ON if E2 turns ON. Even if E2 turns OFF, coil E1 stays ON until coil E1(R) is energized by E3.

NOTE: If both E2 and E3 were ON at the same time, coil E1 would be OFF. This is because rungs are scanned from top to bottom, so the status of the reset coil in the second rung is the last one to be written to the output table. If the order of the rungs was reversed, the set coil would be the last one scanned, so E1 would be ON if E2 and E3 were both ON at the same time.



Note

When the level of coil checking is SINGLE, you can use a specific %M or %Q reference with only one Coil, but you can use it with one SET Coil and one RESET Coil simultaneously. When the level of coil checking is WARN MULTIPLE or MULTIPLE, then each reference can be used with multiple Coils, SET Coils, and RESET Coils. With multiple usage, a reference could be turned ON by either a SET Coil or a normal Coil and could be turned OFF by a RESET Coil or by a normal Coil.

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.

Retentive SET coils write an undefined result to the transition bit for the given reference. (Refer to the information on “Transitions and Overrides” in chapter 2, “System Operation.”)

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.

Retentive RESET coils write an undefined result to the transition bit for the given reference. (Refer to the information on "Transitions and Overrides" in chapter 2, "System Operation.")

Links

Horizontal and vertical links, which appear as straight lines on-screen, 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.

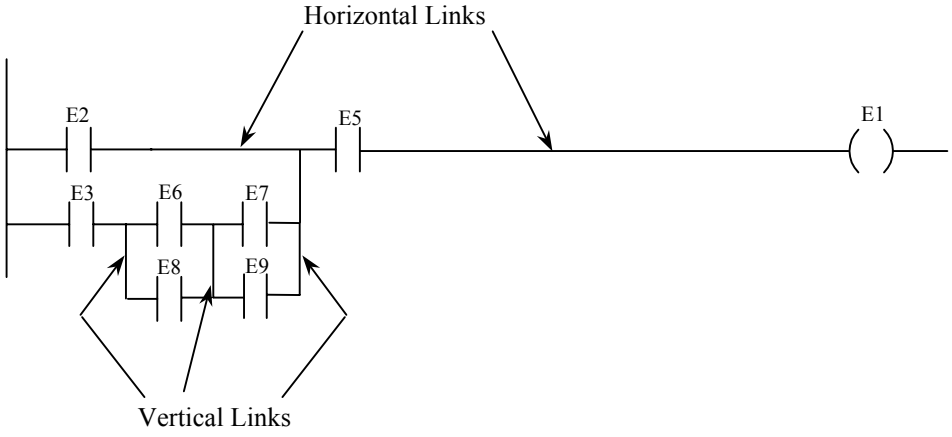
Note

You can not use a horizontal link to tie a function or coil to the left power rail. You can, however, use %S7, the AWL_ON (always on) system bit with a normally open contact tied to the power rail to call a function every sweep.

Example

Several links are used in the following example:

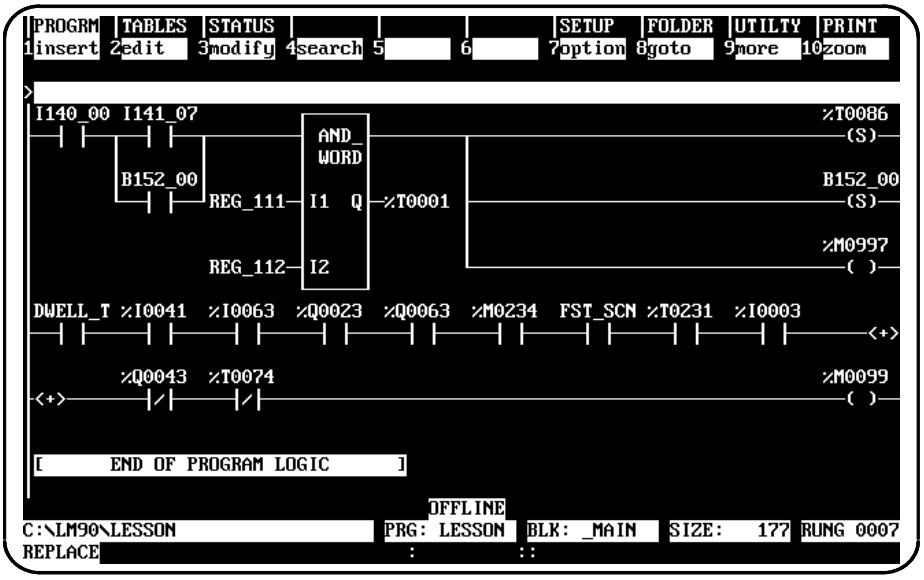
- Horizontal links connect contact E2 to contact E5, and contact E5 to coil E1.
- Vertical links connect contact E8 across contact E6, contact E9 across contact E7, and the right side of contacts E7/E9 to the junction of contacts E2 and E5.



Continuation Coils (———<+>) and Contacts (<+>———)

Continuation coils (———<+>) and continuation contacts (<+>———) are used to continue relay ladder rung 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. There needs to be a continuation coil before the logic executes a continuation contact. 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. An example continuation coil and contact are shown below:



Chapter 5 Timers and Counters

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

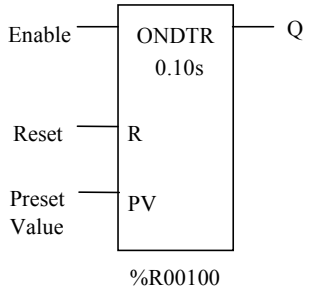
Abbreviation	Function	Page
ONDTR	Retentive On-Delay Timer	5-3
TMR	Simple On-Delay Timer	5-5
OFDT	Off-Delay Timer	5-8
UPCTR	Up Counter	5-11
DNCTR	Down Counter	5-13

Function Block Data Required for Timers and Counters

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

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

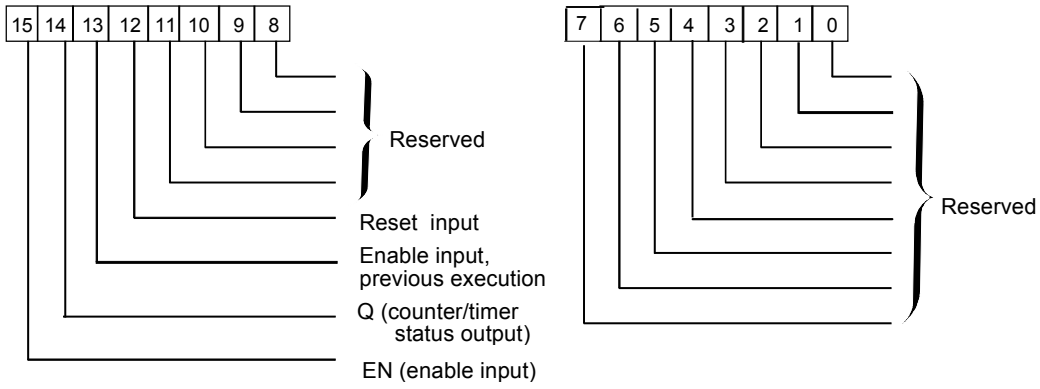
When you enter a timer or counter, you must enter a beginning address (the address for word 1) for this three-word block directly below the graphic representing the function. In the following example, this beginning address is %R00100.



Note

Make sure that the addresses in the three-word block are not used elsewhere in your program (this duplicate use is called “overlapping”). Logicmaster does ***not*** check or warn you if register blocks overlap. Timers and counters will not work correctly if you overlap their three-word blocks.

The control word (the third word in the three-word block) stores the state of the Boolean inputs and outputs of its associated function block, as shown in the following format:



Bits 0 through 11 are reserved by the PLC for use in maintaining timer accuracy; these bits (0 through 11) are not used for counter function blocks.

Note

Use care if you use the same address for the function's PV (Preset Value) input parameter as the second word in the three-word block. If PV is not a constant, the PV input normally is addressed to a different memory location than the second word. Some programmers choose to use the second word address for the PV input, such as using %R0102 when the three-word block starts at %R0101. This allows an application to change the PV while the timer or counter is running. Applications can read the first (CV) or third (Control) words, but the application cannot write to these values, because if they were written to, the function would not work.

Special Note on Certain Bit Operations

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15 as shown above.

ONDTR

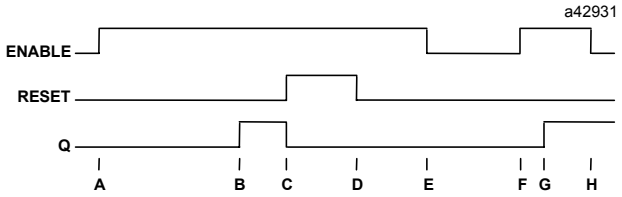
A retentive on-delay timer (ONDTR) increments while it receives power flow and holds its value when power flow stops. Time may be counted in tenths of a second (the default selection), hundredths of a second, or thousandths of a second. The range is 0 to +32,767 time units; therefore, the timing range is 0.001 to 3,276.7 seconds. 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 timers will be the same.

When the current value 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.

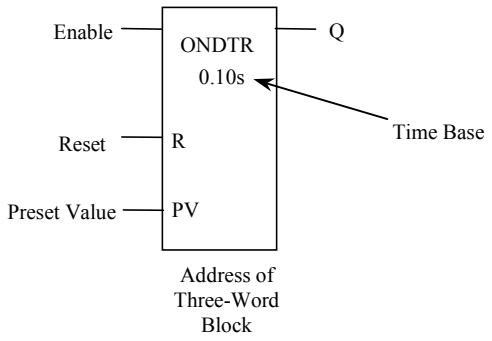
E = ENABLE goes low; timer stops accumulating. Accumulated time stays the same.

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 CV 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. On 35x, 36x, and 37x series PLCs, if the enable to the ONDTR is low, PV = 0 and reset R receives power-flow, then the output will be low. However, on the 311–341 PLCs, under these same conditions, the output will be high.



Parameters

Parameter	Description
Address of Three-Word Block	The ONDTR uses three consecutive words (registers) of %R memory to store the following: • 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 the first of three consecutive words (registers) directly below the graphic representing the function (the use of the other two words is implied). **Caution:** Do not write to these three words with other instructions. Overlapping these references will result in erratic operation of the timer.
Enable	When enable receives power flow, the timer begins functioning.
R	Reset input. When R receives power flow, it resets the current value to zero. Input R, if used, must be connected by one or more contacts to the power rail. This requires that the ONDTR instruction be placed in the first position (left-most position) in the rung.
PV	Preset Value input. PV is the value to copy into the timer's preset value when the timer is enabled or reset. The timer will turn on the Q output when it times to the PV value.
Q	Output Q is energized when the current value (CV) is greater than or equal to the preset value (PV).
Time Base	This parameter may be programmed for time increment of tenths (0.1), hundredths (0.01), or thousandths (0.001) of seconds. This time base value is multiplied by the number in the Preset Value (PV) input parameter to determine the actual preset value.

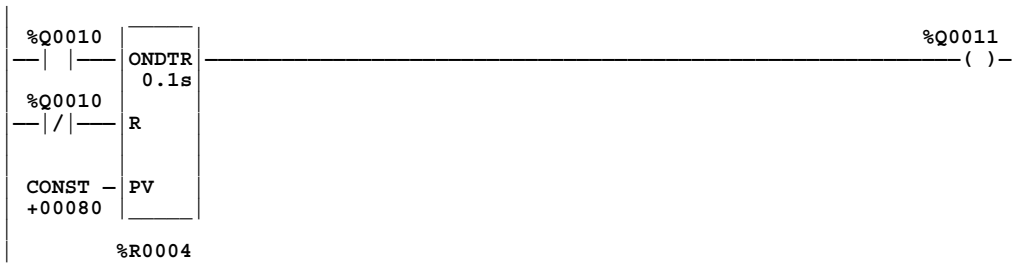
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

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

Example

In the following example, a retentive on-delay timer is used to produce an output (%Q0011) that turns on 8.0 seconds after %Q0010 turns on, and turns off when %Q0010 turns off. This is because when %Q0010 turns off, its normally closed contact passes power to the reset (R) input. The 8.0 second time value is obtained by multiplying the PV value (80) times the time base value (0.1s).



TMR

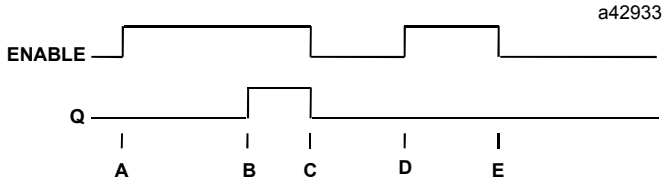
The simple on-delay timer (TMR) function increments while it receives power flow and resets to zero when power flow stops. Time may be counted in tenths of a second (the default selection), hundredths of a second, or thousandths of a second. The range is 0 to +32,767 time units, therefore the timing range is 0.001 to 3,276.7 seconds. 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 the timer has been enabled since it 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 timers will be the same.

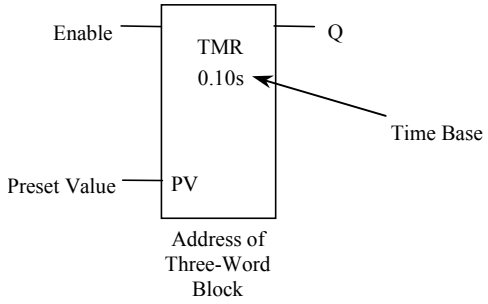
The timer's elapsed time value (CV - current value) continues to accumulate as long as the enabling logic remains ON. When the current value (CV) equals or exceeds the preset value (PV), the function begins passing power flow to the right. The timer continues accumulating time until the maximum value (32,767 time units) is reached. When the enabling input 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 = Current value reaches preset value PV; Q goes high, and timer continues accumulating time.

C = ENABLE goes low; Q goes low; timer stops accumulating time and current time is cleared.
D = ENABLE goes high; timer starts accumulating time.
E = ENABLE goes low before current value reaches preset value PV; Q remains low; timer stops accumulating time and is cleared to zero.



Parameters

Parameter	Description
Address of Three-Word Block	The TMR uses three consecutive words (registers) of %R memory to store the following: • 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 the first of three consecutive words (registers) directly below the graphic representing the function (the use of the other two words is implied). **Caution:** Do not write to these three words with other instructions. Overlapping these references will result in erratic operation of the timer.
Enable	When enable receives power flow, the timer begins functioning. When the enable input goes off, the current value is reset to zero and Q is turned off.
PV	Preset Value input. PV is the value to copy into the timer's preset value when the timer is enabled or reset. The timer will turn on the Q output when it times to the PV value.
Q	Output Q is energized when the current value (CV) is greater than or equal to the preset value (PV).
Time Base	This parameter may be programmed for time increment of tenths (0.1), hundredths (0.01), or thousandths (0.001) of seconds. This time base value is multiplied by the number in the Preset Value (PV) input parameter to determine the actual preset value.

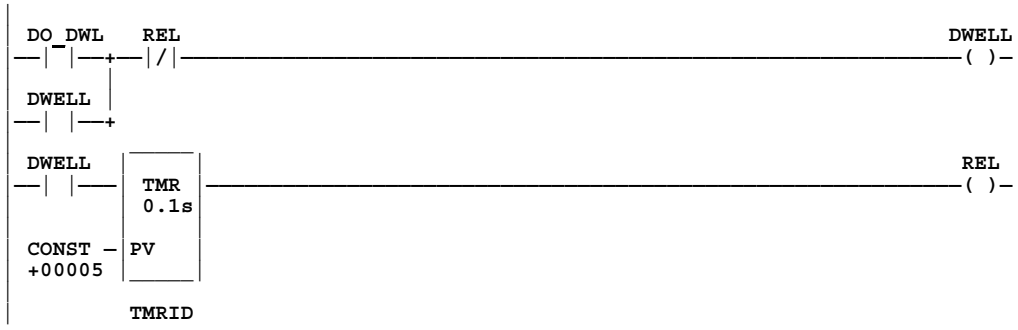
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

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

Example

In the following example, a TMR timer is used to control the length of time that coil DWELL is on. The timing process starts when the normally open (momentary) contact DO_DWL turns on, which turns on coil DWELL. A DWELL contact keeps coil DWELL energized ("latched") when contact DO_DWL opens; also, another DWELL contact enables the timer. When the timer reaches its preset value of one-half second, coil REL energizes. The normally closed REL contact opens, interrupting the latched-on condition of coil DWELL, which turns off. The DWELL contact on the timer's enable input opens, which interrupts power flow to the timer, resets its current value, and de-energizes coil REL. The circuit is then ready for another activation of contact DO_DWL.



OFDT

The off-delay timer's (OFDT) accumulated value increments while power flow is off, and resets to zero when power flow is on. Time may be counted in tenths of a second (the default selection), hundredths of a second, or thousandths of a second. The range is 0 to +32,767 time units, which gives a range of .001 to 3,276.7 seconds. 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 OFDT uses word 1 [register] as its CV storage location—see the "Parameters" section on the next page for additional information.) The output remains on as long as the function receives power flow. If the function stops receiving power flow from the left, its output remains on temporarily, and the timer starts accumulating time in the current value; once the accumulated value reaches the preset value, the output turns off.

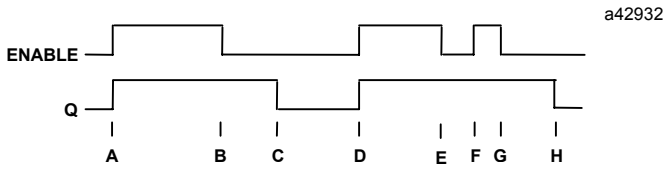
Note

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

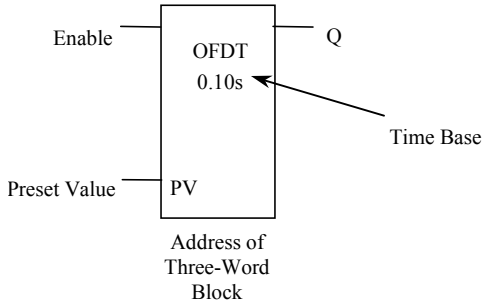
The OFDT does not pass power flow if the preset value is zero or negative.

Each time the function is invoked by turning off the enabling logic (at the enable input), 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. When that occurs, the timer stops accumulating time—see Part C below.

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



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



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 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 of Three-Word Block	The OFDT timer uses three consecutive words (registers) of %R memory to store the following: • 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 the first of three consecutive words (registers) directly below the graphic representing the function (the use of the other two words is implied). **Caution:** Do not write to these three words with other instructions. Overlapping these references will result in erratic operation of the timer.
Enable	While the enable input is on, output Q stays on, and the current value (CV) is held to zero. When the enable input turns off, the timer begins timing. When the current value (CV) reaches the preset value (PV), the timer stops timing, and Q turns off.
PV	Preset Value input. PV is the value to copy into the timer's preset value when the timer is enabled or reset. The timer will turn off the Q output when it times to the PV value.
Q	Output Q is energized (1) when the enable input is on and (2) while the current value (CV) is less than the preset value (PV) after the enable input turns off.
Time Base	This parameter may be programmed for time increment of tenths (0.1), hundredths (0.01), or thousandths (0.001) of seconds. This time base value is multiplied by the number in the Preset Value (PV) input parameter to determine the actual preset value.

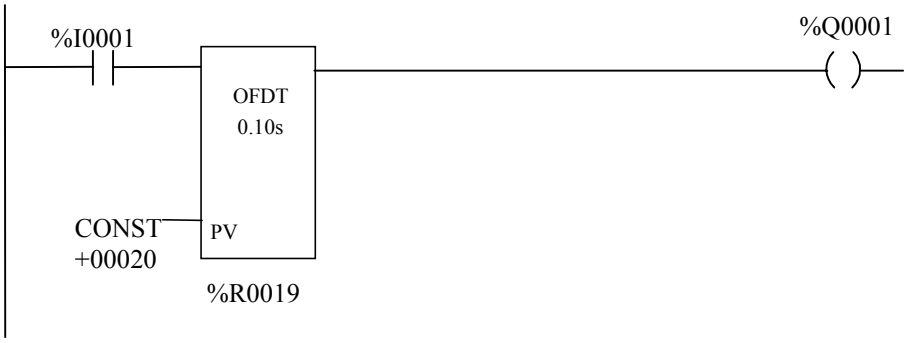
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
PV	•	•	•	•	•		•	•	•	•	•	•
Q	•											•

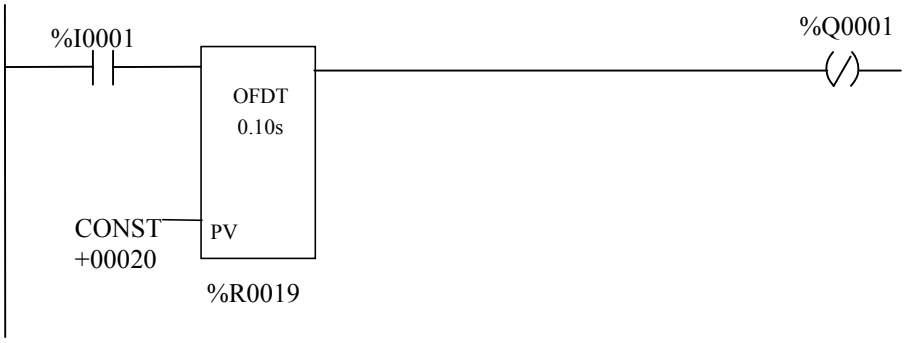
• Valid reference or place where power may flow through the function.

Examples

In the following example, an OFDT timer turns on output coil %Q0001 whenever contact %I0001 is closed. After %I0001 opens, %Q0001 stays on for 2 seconds then turns off.



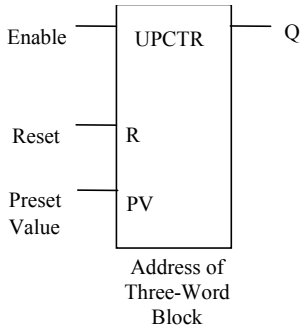
In the next example, the output action is reversed by the use of a negated output coil. In this circuit, an OFDT timer turns off negated output coil %Q0001 whenever contact %I0001 is closed. After %I0001 opens, %Q0001 stays off for 2 seconds then turns on.



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 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 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 state of the UPCTR is retentive on power failure; no automatic initialization occurs at power-up.



Parameters

Parameter	Description
Address of Three-Word Block	The UPCTR Up Counter uses three consecutive words (registers) of %R memory to store the following: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter a UPCTR, you must enter an address for the location of the first of three consecutive words (registers) directly below the graphic representing the function (the use of the other two words is implied). **Caution:** Do not write to these three words with other instructions. Overlapping these references will result in erratic operation of the timer.
Enable	On each positive transition (off to on) of the enable input, the current count value (CV) is incremented by one.
PV	Preset Value input. PV is the value copied into the counter's preset value when the counter is enabled or reset. The counter will turn on the Q output when it counts up to the PV value. If the preset value is a constant, it must be a positive number between 0 and 32,767.
Q	Output Q is energized when the current count value (CV) is greater than or equal to the preset value (PV).
R	Reset Input. When the R input turns on, the current count value (CV) is reset to zero.

Valid Memory Types

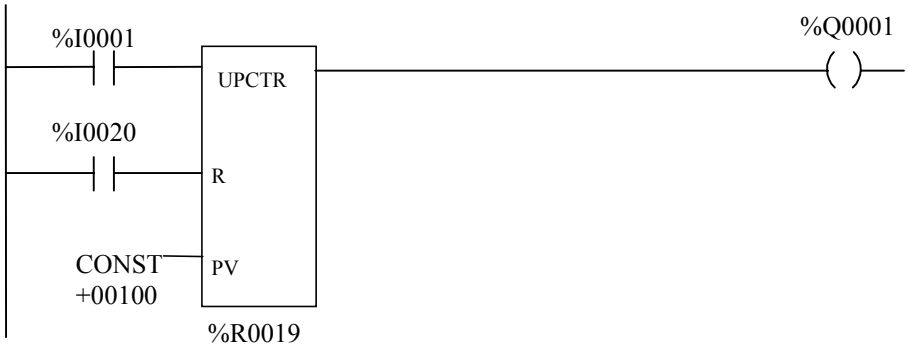
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

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

Examples

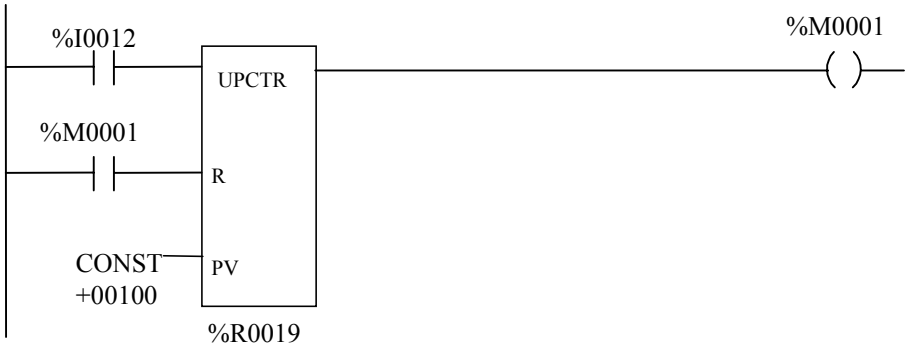
Basic Counter Circuit

In the following example, the UPCTR will increment its current count value (CV) by one each time %I0001 transitions from off to on. The PV input sets the preset value to 100 counts. When the counter counts to 100, coil %Q0001 will be turned on. The counter will continue to count %I0001 transitions beyond its preset value (of 100) until it either reaches its maximum count value (32, 767), or until %I0020 closes and resets the counter. %Q0001 will be on anytime the CV value is equal to or greater than the PV value.



Self-Resetting Counter Circuit

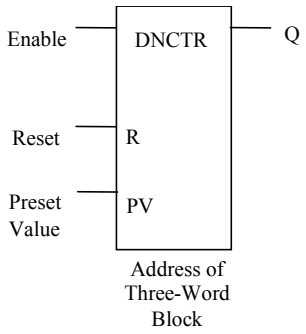
In the next example, every time input %I0012 transitions from OFF to ON, the UPCTR counter counts up by 1. Coil %M0001 is energized whenever 100 %I0012 transitions have been counted. Once %M0001 turns ON, the accumulated count is reset to zero by the %M0001 contact on the R input, and %M0001 will turn off.



DNCTR

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

The current value of the DNCTR is retentive on power failure; no automatic initialization occurs at power-up.



Parameters

Parameter	Description
Address of Three-Word Block	The DNCTR Down Counter uses three consecutive words (registers) of %R memory to store the following: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter a DNCTR, you must enter an address for the location of the first of three consecutive words (registers) directly below the graphic representing the function (the use of the other two words is implied). **Caution:** Do not write to these three words with other instructions. Overlapping these references will result in erratic operation of the counter.
Enable	On each positive transition (off to on) of the enable input, the current count value (CV) is decremented by one.
PV	Preset Value input. PV is the value copied into the counter's preset value (PV) and current value (CV) registers when the counter is enabled or reset. The counter will turn on the Q output when it counts down from the current value to zero.
Q	Output Q is energized when the current count value (CV) is less than or equal to zero.
R	Reset Input. When the R input turns on, the current count value (CV) is reset to the preset value (PV, and output Q is turned off.

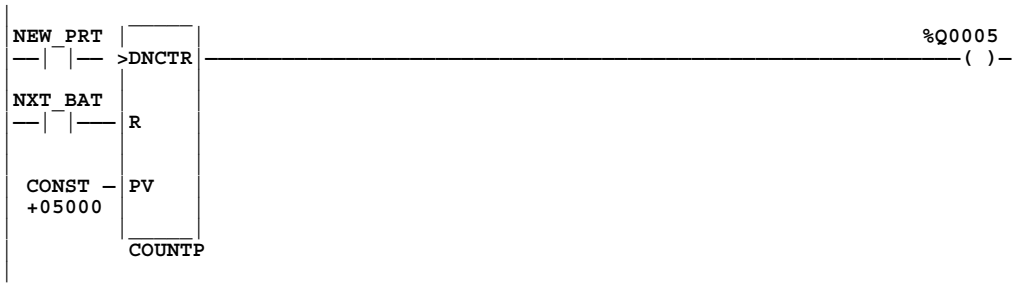
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

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

Examples

In the following example, the down counter identified as COUNTP counts 5000 new parts before energizing output %Q0005.

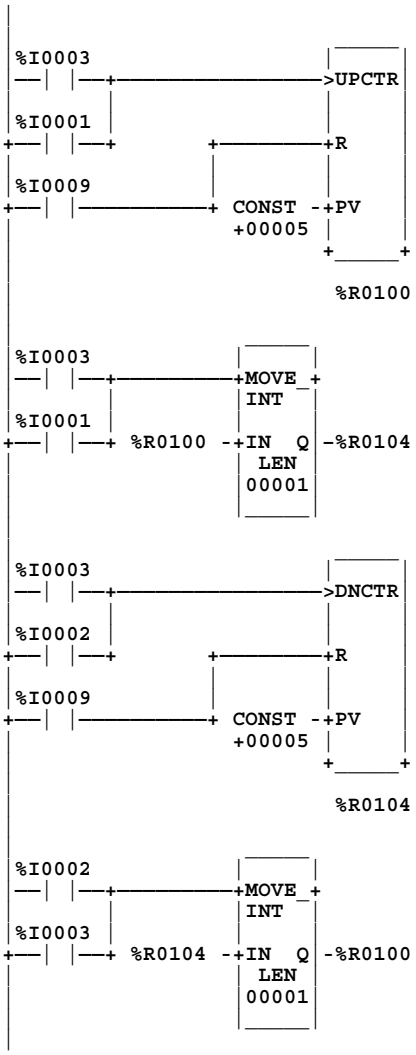


Inventory Count Examples

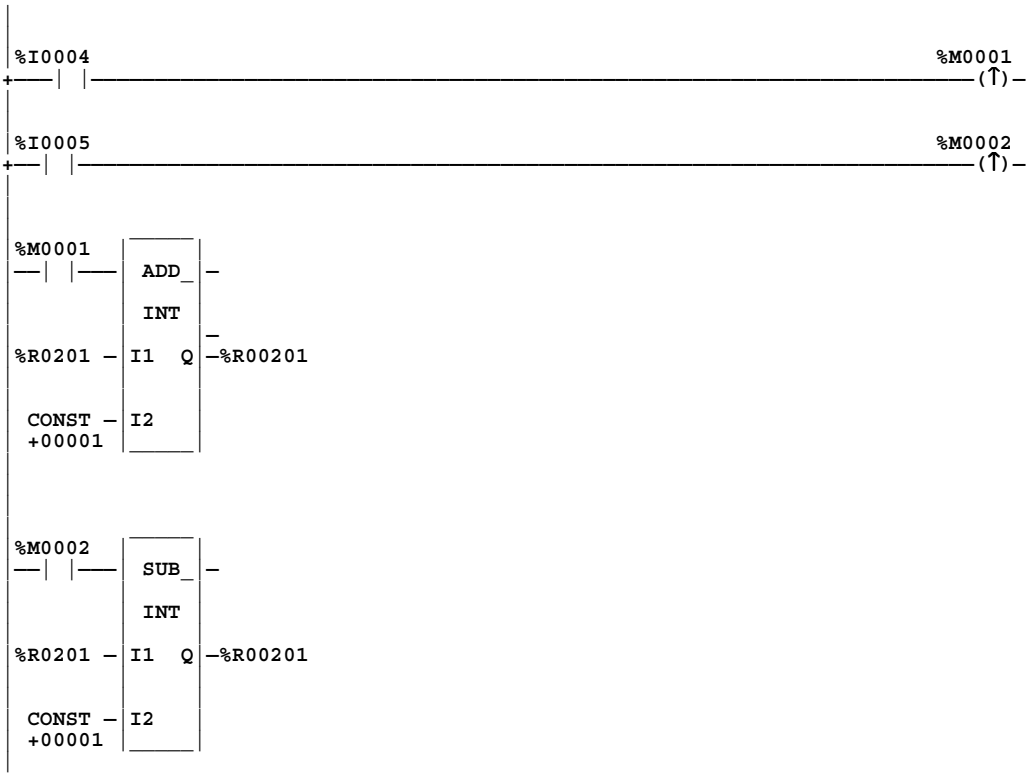
In the next example, the PLC is used to keep track of the number of parts contained in a temporary storage area. There are two ways of accomplishing this function using the Series 90-30/20/Micro instruction set.

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 (%I0001 closes), 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 (%I0002 closes), decreasing the inventory storage value by 1. To avoid conflict with the shared register, both counters use different register addresses. When a register counts, its current value is moved to the current value register of the other counter.

In the following example, %I0001 increments the count, %I0002 decrements the count, %I0009 resets the count to zero, and %I0003, when on, holds the count at its current value regardless of what %I0001 and %I0002 do. The count value can be read from %R0100.



A second method to provide storage tracking, shown below, uses ADD and SUB functions that share a common register, %R00201, on their outputs. When the count increases ($I0004 closes), the ADD instruction increments the value in %R00201. When the count decreases (%I0005 closes), the SUB instruction decrements the value in %R00201. In this case, transition coils are used to provide "one-shot" inputs to the ADD and SUB functions. If the enable inputs were not one-shot types, the ADD and SUB functions would execute once for every scan that they were enabled. (Transition coils are not needed with UPDTR and DNCTR functions since their enable inputs have a built-in transition function.) See Chapter 6 for details about the ADD and SUB functions.



Chapter 6 Math Functions

This chapter describes the math functions of the Series 90-30/20/Micro Instruction Set:

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.	6-7
SQRT	Square Root	Find the square root of an integer or real value.	6-9
SIN, COS, TAN, ASIN, ACOS, ATAN	Trigonometric Functions †	Perform the appropriate function on the real value in input IN.	6-11
LOG, LN EXP, EXPT	Logarithmic/Exponential Functions †	Perform the appropriate function on the real value in input IN.	6-13
RAD, DEG	Radian Conversion †	Perform the appropriate function on the real value in input IN.	6-15

† Trigonometric Functions, Logarithmic/Exponential Functions, and Radian Conversion functions are only available on the model 35x and 36x series CPUs, Release 9.00 or later, and on all releases of CPU352 and CPU37x.

Note

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

Standard Math Functions (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.

Rules for Math Functions

Sign of Result	Standard math rules for signed number arithmetic apply to determining the sign of the result.
Addition	The ADD instruction uses the formula I1 + I2 = Q.
Subtraction	The SUB instruction uses the formula I1 – I2 = Q**.**
Multiplication	The MUL instruction uses the formula I1 x I2 = Q.
Division	The DIV instruction uses the formula I1 ÷ I2 = Q. **For INT and DINT types**. DIV rounds down to a whole number quotient (any remainder is discarded) for the INT or DINT types; it does not round to the closest integer. For example, 53 divided by 5 = 10 (the remainder of 3 is discarded). **For REAL type**. DIV produces a decimal number result for the Real type
Modulo Division	The MOD instruction can only use types INT and DINT (REAL is not supported). The MOD instruction uses the formula I1 ÷ I2 = Q. However, MOD produces only the remainder from the division operation and discards the quotient. For example, 53 divided by 5 = 3 (the quotient of 10 is discarded).

Data Types for Math Functions

After you have programmed a math function, you can select the data type. The data type will appear on the function just below the function's name (see example in next figure). The three data types available for math functions are listed in the following table:

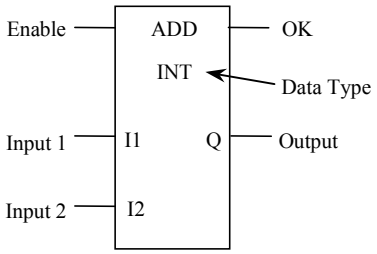
Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
REAL*	Floating Point

*REAL data type is only available on 35x and 36x series CPUs, Firmware Release 9.00 or later, and on all releases of CPU352 and CPU37X.

The default data type is signed integer. For more information on data types, please refer to Chapter 2, Section 2, "Program Organization and User References/Data."

If the operation of INT or DINT 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 the operation does not result in overflow (and the inputs are valid numbers), the ok output is set ON; otherwise, it is set OFF. If signed or double precision integers are used, the sign of the result depends on the signs of inputs I1 and 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 overflow, unless an invalid operation occurs.
Q	Output Q contains the result of the operation.

Valid Memory Types

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

• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT or REAL.
† When using Logicmaster, you will only be able to enter values between –32,768 and +32,767 for double precision signed integer operations. With VersaPro, you can enter full double precision values.

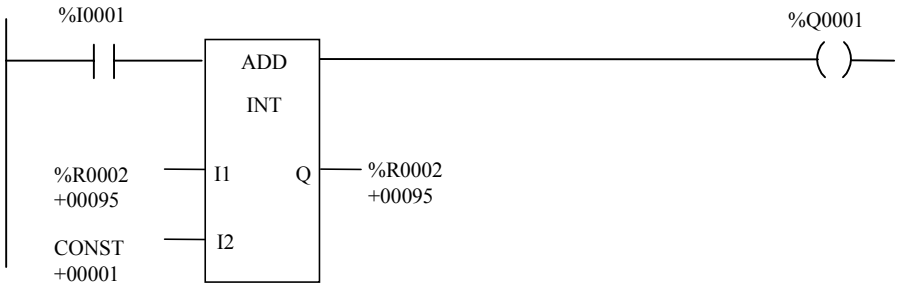
Note

The default type is INT for 16-bit or single register operands. In Logicmaster, press **F10** to change the Types selection to DINT, 32-bit double word, or REAL (for the 35x, 36x, and 37x series CPUs only). PLC INT values occupy a single 16-bit register, %R, %AI or %AQ. DINT values require two consecutive registers with the low 16 bits in the first word and the upper 16 bits with the sign in second word. REAL values, in the 35x and 36x series CPU (Release 9.00 or later) and all releases of CPU352 and CPU37x, also occupy a 32-bit double register with the sign in the high bit followed by the exponent and mantissa.

Math Function Examples

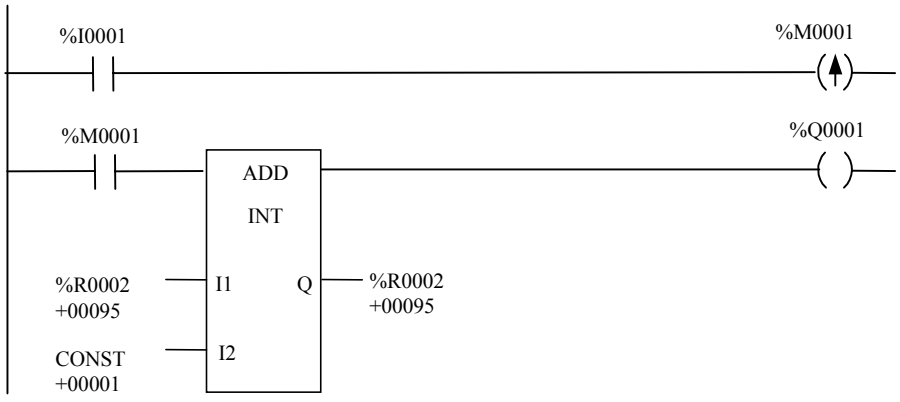
ADD Circuit with a Problem

In the following example, an attempt was made to create a counter circuit that would count the number of times switch %I0001 closes. The running total is stored in register %R0002. The intent of this design is that when %I0001 closes, the ADD instruction adds one to the value in %R0002 (the input on I2) and places the new value right back into %R0002 (the output on Q). The problem with this design is that the ADD instruction will execute once every PLC scan while %I0001 is closed. So, for example, if %I0001 stays closed for five scans, the output will increment five times, even though %I0001 only closed once during that period. To correct this problem, the enable input to the ADD instruction should come from a transition ("one-shot") coil, as shown in the second figure below.



In the following improved circuit, the %I0001 input switch controls a transition ("one-shot") coil, %M0001, whose contact turns on the enable input of the ADD function for only one scan each time contact %I0001 closes. In order for the %M0001 contact to close again, contact %I0001 has to open and close again.

Corrected ADD Circuit Design



Math Functions and Data Types

Function	Operation	Displays as
ADD INT	Q(16 bit) = I1(16 bit) + I2(16 bit)	5-digit base 10 number with sign
ADD DINT	Q(32 bit) = I1(32 bit) + I2(32 bit)	8-digit base 10 number with sign
ADD REAL*	Q(32 bit) = I1(32 bit) + I2(32 bit)	7-digit base 10 number, sign and decimal
SUB INT	Q(16 bit) = I1(16 bit) – I2(16 bit)	5-digit base 10 number with sign
SUB DINT	Q(32 bit) = I1(32 bit) – I2(32 bit)	8-digit base 10 number with sign
SUB REAL*	Q(32 bit) = I1(32 bit) – I2(32 bit)	7-digit base 10 number, sign and decimal
MUL INT	Q(16 bit) = I1(16 bit) * I2(16 bit)	5-digit base 10 number with sign
MUL DINT	Q(32 bit) = I1(32 bit) * I2(32 bit)	8-digit base 10 number with sign
MUL REAL*	Q(32 bit) = I1(32 bit) * I2(32 bit)	7-digit base 10 number, sign and decimal
DIV INT	Q(16 bit) = I1(16 bit) / I2(16 bit)	5-digit base 10 number with sign
DIV DINT	Q(32 bit) = I1(32 bit) / I2(32 bit)	8-digit base 10 number with sign
DIV REAL*	Q(32 bit) = I1(32 bit) / I2(32 bit)	7-digit base 10 number, sign and decimal

* 35x and 36x series CPUs only, Release 9 or later, and all releases of CPU352 and CPU37x.

Note

The input and output data types must be the same for math functions. The MUL and DIV functions do not support a mixed mode as the Series 90-70 PLCs do. For example, the MUL INT of two 16-bit inputs produces a 16-bit product, not a 32-bit product. Using MUL DINT for a 32-bit product requires both inputs to be 32-bit. The DIV INT divides a 16-bit I1 by a 16-bit I2 for a 16-bit result, while DIV DINT divides a 32-bit I1 by 32-bit I2 for a 32-bit result.

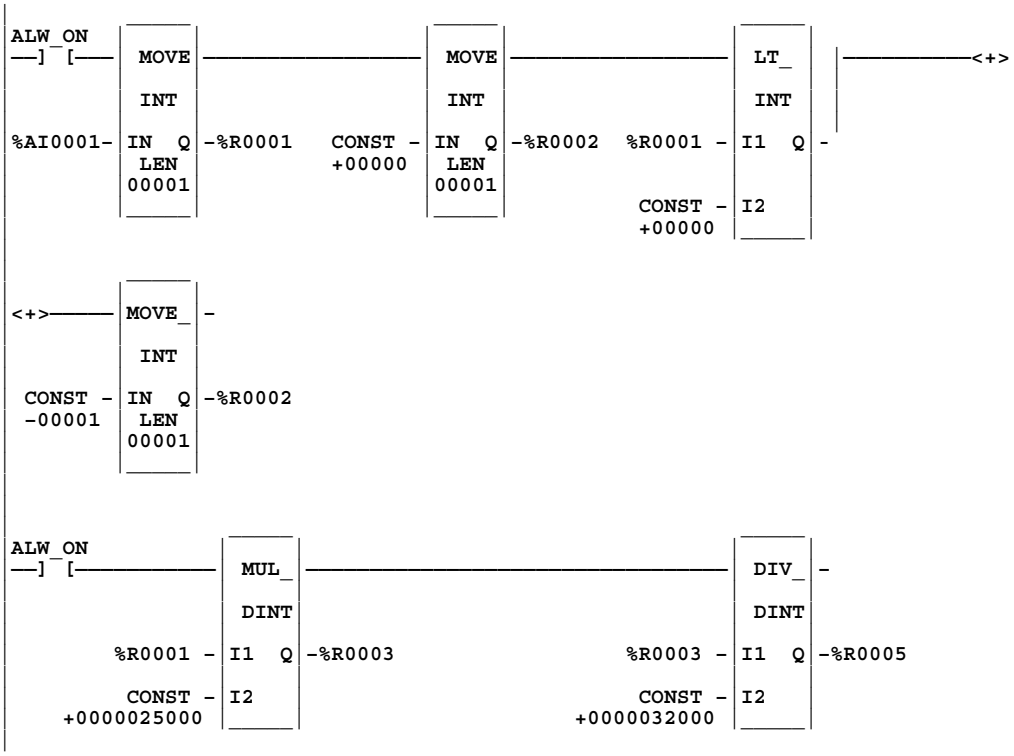
When enabled, these functions pass power if there is no math overflow. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

Be careful to avoid overflows when using MUL and DIV functions. If you have to convert INT to DINT values, remember that the CPU uses standard 2's complement with the sign extended to the highest bit of the second (most significant) word. You must check the sign of the low 16-bit word and extend it into the second 16-bit word. If the most significant bit in a 16-bit INT low word is 0 (indicating positive value), move a 0 to the second word. If the most significant bit in a 16-bit word is 1 (indicating a negative value), move a –1 or hex 0FFFFh to the second word. Converting from DINT to INT is easier as the low 16-bit word (first register) is the INT part of a DINT 32-bit word. The upper 16 bits or second word should be either a 0 (positive) or –1 (negative) value or the DINT number is too big to convert to 16 bit.

Example

A common application is to scale analog input values with a MUL operation followed by a DIV and possibly an ADD operation. A 0 to ± 10 volt analog input will place values of 0 to ± 32,000 in its corresponding %AI input register. Multiplying this input register using an INT MUL function will result in an overflow since an INT type instruction has an input and output range of 32,767 to –32,768. Using the %AI value as in input to a MUL DINT will also not work as the 32-bit I1 will combine 2 analog inputs at the same time. To solve this problem, you can move the analog input to the low word of a double register, then test the sign and set the second register to 0 if the sign tests positive or –1 if negative. Then use the double register just created with a MUL DINT which gives a 32-bit result, and which can be used with a following DINT DIV function.

For example, the following logic could be used to scale a ± 10 volt input %AI1 to ± 25000 engineering units in %R5.



An alternate, but less accurate, way of programming this circuit using INT instructions involves placing the DIV instruction first, followed by the MUL instruction. The value of I2 for the DIV instruction would be 32, and the value of I2 for the MUL would be 25. This maintains the scaling proportion of the above circuit and keeps the values within the working range of the INT type instructions. However, the DIV instruction inherently discards any remainder value, so when the DIV output is multiplied by the MUL instruction, the error introduced by a discarded remainder is multiplied. The percent of error is non-linear over the full range of input values and is greater at lower input values.

By contrast, in the example above, the results are more accurate because the DIV operation is performed last, so the discarded remainder is not multiplied. If even greater precision is required, substitute REAL type math instructions in this example so that the remainder is not discarded.

MOD (INT, DINT)

The Modulo (MOD) function is used to divide one value by another value of the same data type to obtain the remainder. The sign of the result is always the same as the sign of input parameter I1.

The MOD function operates on these types of data:

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

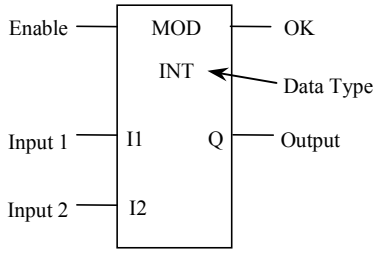
The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, please refer to chapter 2, section 2, "Program Organization and User References/Data."

When the function 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.



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 overflow.
Q	Output Q contains the remainder, if any, that results from dividing I1 by I2. If the value in I2 is an even multiple of I1, output Q will be zero, indicating no remainder.

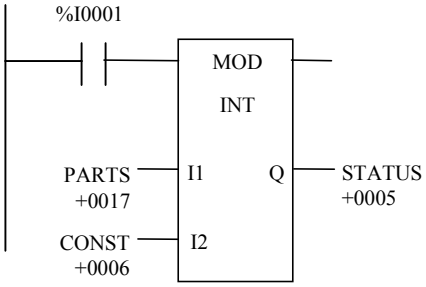
Valid Memory Types

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

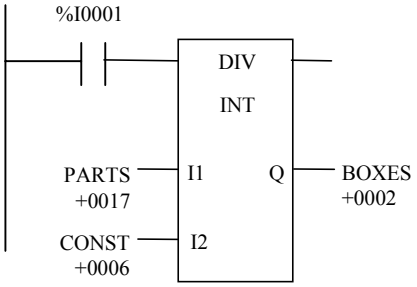
• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT.
† Constants are limited to values between –32,768 and +32,767 for double precision signed integer operations.

Example

In the following example, boxes are being automatically filled with parts. One box holds six parts. This circuit determines the status of the current box being filled by using modulo division. When enabled, the MOD function divides the register (PARTS) holding the count of parts produced, by six. The output (STATUS) of the MOD instruction indicates how many parts (between 1 and 5) have been loaded into the current box. When the current box is full, the output at Q will equal zero; if the current box is only partially filled, the output will indicate the number of parts already in the box. The values in the example show that a total 17 parts have been produced and that the current box has five parts in it. (The other 12 parts filled two boxes.)



To determine the number of boxes filled, you could use the DIV instruction in the following circuit.



One possible problem with these circuits is that the register nicknamed PARTS can only hold a maximum of 32,767 counts. If you need to count higher than that, some additional logic will be required to (1) reset the PARTS register before it reaches maximum, (2) to capture the number of boxes filled before you reset the PARTS register, and (3) to reset the PARTS register when the STATUS register is zero so that its count stays accurate.

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.

For data types INT and DINT, only the whole number portion of the square root will be output. The fractional portion will be dropped. For example, the square root of 2 or 3 will be 1, and the square root of 5, 6, 7, or 8 will be 2.

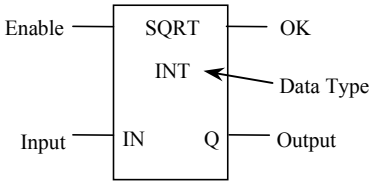
Note

The REAL data type is only available on 35x and 36x series CPUs, Release 9.00 or later, and on all releases of CPU352 and CPU37x.

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."

OK is set ON if the function is performed without overflow. If one of the following invalid operations occurs, OK is set OFF:

- IN < 0
- IN is NaN (Not a Number)



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.
Q	Output Q contains the square root of IN. However, for INT and DINT, only the whole number portion will be kept; any fractional portion will be discarded.

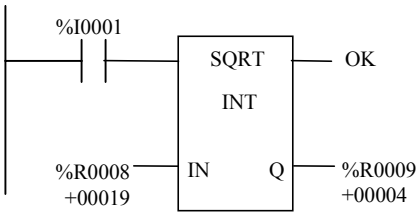
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		o	o	o	o		o	•	•	•	•†	
ok	•											•
Q		o	o	o	o		o	•	•	•		

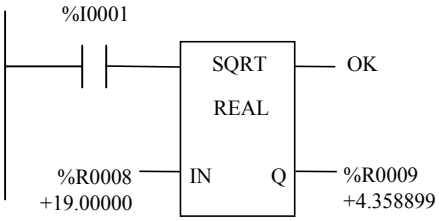
• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT and REAL.
† Constants are limited to values between –32,768 and +32,767 for double precision signed integer operations.

Examples

In the following example, the square root of the integer number located at %R0008 is placed into the %R0009 register whenever %I0001 is ON.



As an alternative to the previous example, the same function can be performed with a REAL-type SQRT instruction, which gives a more precise result as shown in the next figure.



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 designated function on the value at the IN input, 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 $-1 \le IN \le 1$. Given a valid value for the IN parameter, the ASIN_REAL function will produce a result Q such that:

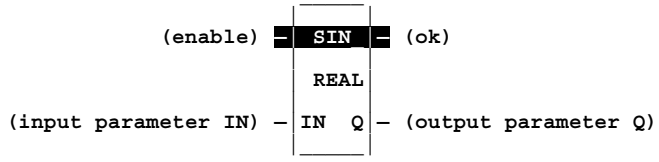
$$\text{ASIN (IN)} = -\frac{\pi}{2} \le Q \le \frac{\pi}{2}$$

The ACOS_REAL function will produce a result Q such that:

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

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

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



Note

The TRIG functions are only available on the 35x and 36x series CPUs, Release 9 or later, and on all releases of CPU352 and CPU37x.

Parameters

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the constant or reference 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.

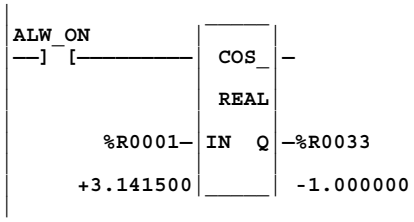
Valid Memory Types

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

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

Example

In the following example, the COS of the value in %R0001 is placed in %R0033.

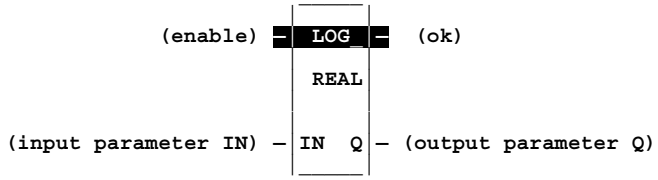


Logarithmic/Exponential Functions (LOG, LN, EXP, EXPT)

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. (NOTE: ***e*** is a constant used in logarithmic calculations. It has an approximate value of 2.71828.)
- For the EXPT function, the value of input I1 is raised to the power specified by the value I2 and the result is placed in output Q. (The EXPT function has three input parameters and two output parameters.)

The ok output will receive power flow, unless IN is NaN (Not a Number) or is negative.



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

The LOG, LN, EXP and EXPT functions are only available on the 35x and 36x series CPUs, Release 9 or later, and on all releases of CPU352 and CPU37x.

Note

When the input value, IN, for the EXP function is negative infinity (-∞), the function returns a value of 0, as expected. In this case, for the CPU352, the function does *not* pass power. For all other 90-30 CPUs, the function *does* pass power, even though the output is 0. (A value of -∞ results from dividing a negative value by zero. It will appear on a Logicmaster screen as –OVERFLOW.)

Valid Memory Types

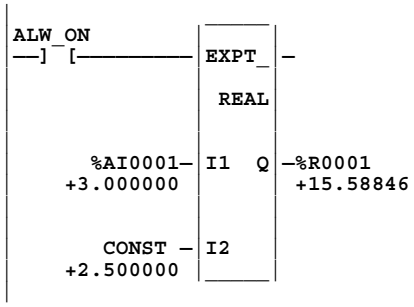
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN*								•	•	•	•	
ok	•											•
Q								•	•	•		
I1*								•	•	•	•	
I2*								•	•	•	•	

* 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.

Example

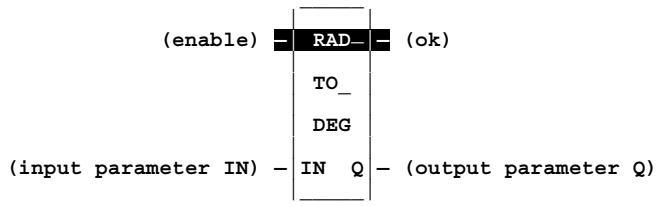
In the following example, the value of %AI0001, +3.000000, is raised to the power of +2.500000, and the result, +15.58846, is placed in %R0001.



Radian Conversion (RAD, DEG)

When the function receives power flow, the appropriate conversion (RAD_TO_DEG or DEG_TO_RAD, i.e., Radian to Degree or vice versa) 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).



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

The Radian conversion functions are only available on the 35x and 36x series CPUs, Release 9 or later, or on all releases of CPU352 and CPU37x.

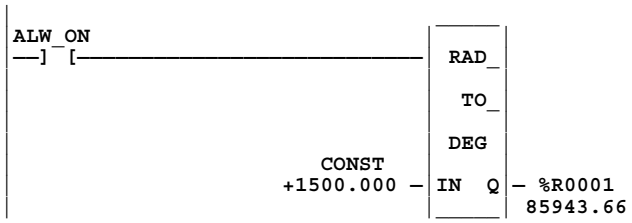
Valid Memory Types

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

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

Example

In the following example, +1500 is converted to DEG and is placed in %R0001.



Chapter 7 Relational Functions

Relational functions are used to determine the relationship of two values. 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
RANGE	Range	Determine whether a number is within a specified range (available for Release 4.5 or higher CPUs).	7-4

Standard Relational Functions (EQ, NE, GT, GE, LT, LE)

When the function receives power flow, it compares input parameter I1 to input parameter I2, which must be of the same data type. Relational functions operate on these types of data:

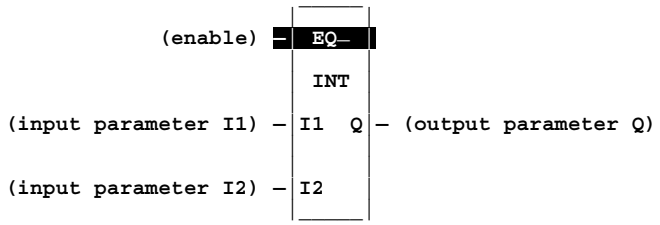
Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
REAL	Floating Point (not available for the RANGE function)

Note

The REAL data type is only available on the 35x and 36x series CPUs, Release 9 or later, and on all releases of CPU352 and CPU37x. The %S0020 system bit is set ON when a relational function using REAL data executes successfully. It is cleared when either input is NaN (Not a Number). The Range function block does not accept REAL type.

The default data type is signed integer. To compare either signed integers, double precision signed integers, or real numbers 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 11, "Conversion Functions") to change the data to one of the supported types.

If input parameters I1 and I2 match the specified relationship, output Q receives power flow and is set ON (1); otherwise, it is set OFF (0).



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).
Q	Output Q is energized when I1 and I2 match the specified relation.

Note

I1 and I2 must be valid numbers, i.e., cannot be NaN (Not a Number).

Expanded Description

Function	Description
Equal	When enabled, if the value at input I1 is equal to the value at input I2, output Q is energized.
Not Equal	When enabled, if the value at input I1 is NOT equal to the value at input I2, output Q is energized.
Greater Than	When enabled, if the value at input I1 is greater than the value at input I2, output Q is energized.
Greater Than or Equal	When enabled, if the value at input I1 is greater than or equal to the value at input I2, output Q is energized.
Less Than	When enabled, if the value at input I1 is less than the value at input I2, output Q is energized.
Less Than or Equal	When enabled, if the value at input I1 is less than or equal to the value at input I2, output Q is energized.

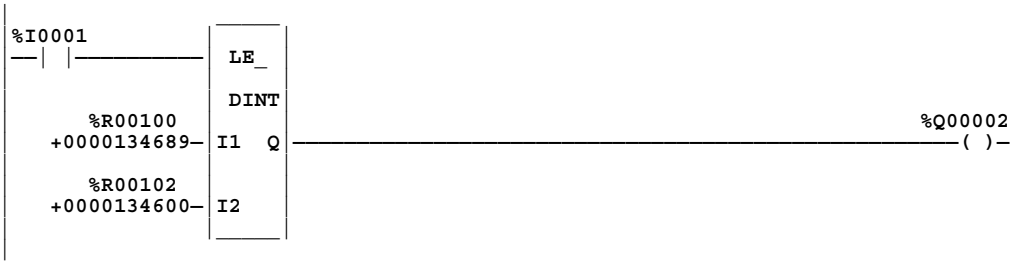
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		o	o	o	o		o	•	•	•	•†	
I2		o	o	o	o		o	•	•	•	•†	
Q	•											•

• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT or REAL.
† Constants are limited to integer values (+32,767 to –32,768) for double precision signed integer operations when programmed with Logicmaster PLC software. When programmed with VersaPro software, full double precision signed integer values are allowed.

Example

In the following example, two double precision signed integers, %R00100/101 and %R00102/103, are compared whenever enable contact %I0001 is on. If the value at input I1 is less than or equal to the value at input I2, coil %Q00002 will be turned on. In the following example, coil %Q00002 is turned off, since I1 is greater than I2.



RANGE (INT, DINT, WORD)

The RANGE function is used to determine if a value is between the range of two numbers.

Note

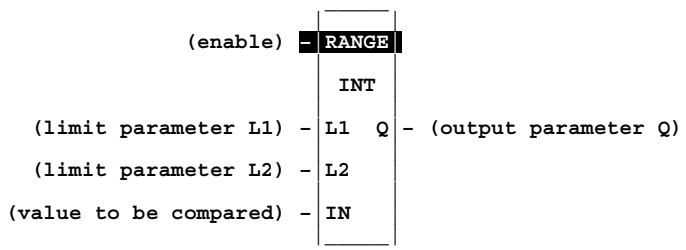
This function is available *only* to Release 4.41 or later CPUs.

The RANGE function operates on these types of data (REAL type is not supported in the RANGE function):

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
WORD	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 limit parameters L1 and L2. When the value is within the range specified by L1 and L2, inclusive, output parameter Q is set ON (1). Otherwise, Q is set OFF (0).



Note

Limit parameters L1 and L2 represent the end points of a range. There is no 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.

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.
Q	Output Q is energized when the value in IN is within the range specified by L1 and L2, inclusive.

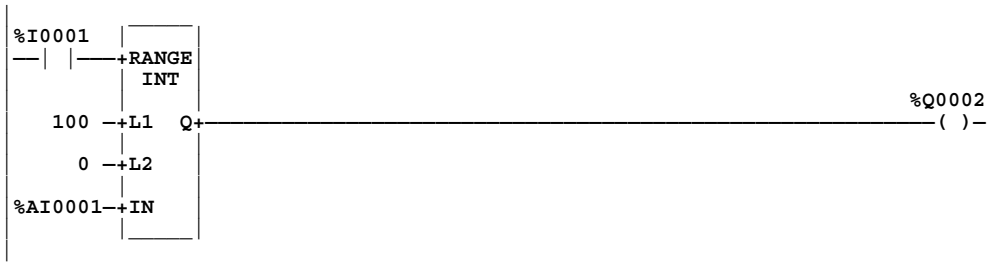
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
L1		o	o	o	o		o	•	•	•	•‡	
L2		o	o	o	o		o	•	•	•	•‡	
IN		o	o	o	o		o	•	•	•		
Q	•											•

• Valid reference or place where power may flow through the function.
o Valid reference for INT or WORD data only; not valid for DINT.
‡ Constants are limited to integer values for double precision signed integer operations.

Example 1

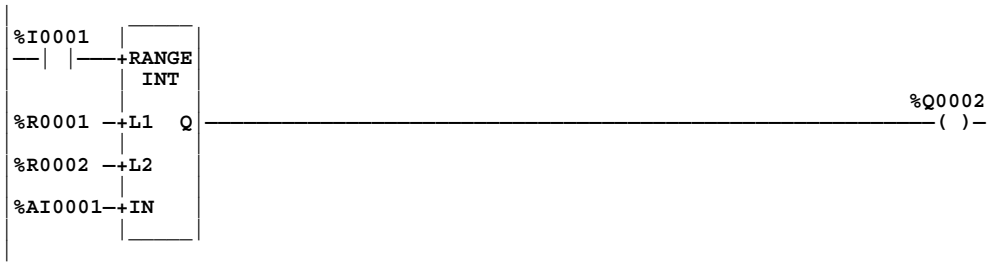
In the following example, %AI0001 is checked to be within a range specified by two constants, 0 and 100.



RANGE Truth Table for Example 1				
Enable State %I0001	L1 Value Constant	L2 Value Constant	IN Value %AI0001	Q State %Q0001
ON	100	0	< 0	OFF
ON	100	0	0 — 100	ON
ON	100	0	> 100	OFF
OFF	100	0	Any value	OFF

Example 2

In this example, %AI0001 is checked to be within a range specified by two register values.



RANGE Truth Table for Example 2				
Enable State %I0001	**L1 Value %R0001**	**L2 Value %R0002**	**IN Value %AI0001**	**Q State %Q0001**
ON	500	0	< 0	OFF
ON	500	0	0 — 500	ON
ON	500	0	> 500	OFF
OFF	500	0	Any value	OFF

Chapter 8 Bit Operation Functions

Bit operation functions perform comparison, logical, and move operations on bit strings. The AND, OR, XOR, and NOT functions are limited to operating on a single word. The remaining bit operation functions may operate on multiple words, with a maximum string length of 256 words. All bit operation functions require the WORD data type.

Although data must be specified in 16-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 %R0100, it would be operated on as 48 contiguous bits.

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

↑

(MSB)

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 and/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.	8-8
SHR	Shift Right	Shift all the bits in a word or string of words to the right by a specified number of places.	8-8
ROL	Rotate Left	Rotate all the bits in a string a specified number of places to the left.	8-10
ROR	Rotate Right	Rotate all the bits in a string a specified number of places to the right.	8-57
BTST	Bit Test	Test a bit within a bit string to determine whether that bit is currently 1 or 0.	8-12
BSET	Bit Set	Set a bit in a bit string to 1.	8-14
BCLR	Bit Clear	Clear a bit within a string by setting that bit to 0.	8-14
BPOS	Bit Position	Locate a bit set to 1 in a bit string.	8-16
MSKCMP	Masked Compare	Compare the contents of two separate bit strings with the ability to mask selected bits (available for Release 4.5 or higher CPUs).	8-18

AND and OR (WORD)

For each scan that it is enabled, an AND or OR function compares the state of each bit in bit string I1 with the corresponding bit in bit string I2, beginning at the least significant bit in each.

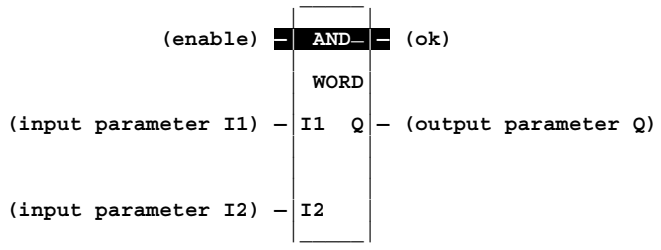
For each two bits compared for the AND function, if both are 1, then a 1 is placed in the corresponding location in output string Q. If either or both bits are 0, then a 0 is placed in string Q in that location.

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 function block. The function is the equivalent of two relay contacts in parallel for each bit position in the string. It can be used to drive indicator lamps directly from input states, or superimpose blinking conditions on status lights.

The function passes power flow to the right whenever power is received.



Parameters

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first word of the first string.
I2	I2 contains a constant or reference for the first word of the second string.
ok	The ok output is energized whenever enable is energized.
Q	Output Q contains the result of the operation.

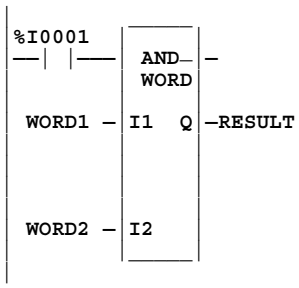
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		•	•	•	•	•	•	•	•	•	•	
I2		•	•	•	•	•	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example

In the following example, whenever input %I0001 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.



WORD1 (I1)	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**
WORD2 (I2)	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**

RESULT (Q)	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**

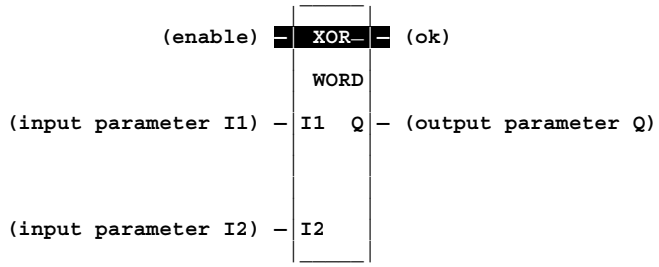
XOR (WORD)

The Exclusive OR (XOR) function is used to compare each bit in the bit string at input I1 with the corresponding bit in the bit string at input I2. If the corresponding bits are different, a 1 is placed in the corresponding position in the output bit string.

The XOR function is useful for comparing two bit strings, or to flash a group of bits on and off at the rate of one ON state per two scans.

For each scan that the XOR is enabled, it compares each bit in string I1 with the corresponding bit in string I2, beginning at the least significant bit in each string. In a comparison, if only one is a logic 1, then a 1 is placed in the corresponding location in bit string Q. 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 switching the enable input to the function at twice the desired rate of flashing; for this application, the enable input should go high for one scan long (use a contact from a one-shot type coil or self-resetting timer circuit).



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.

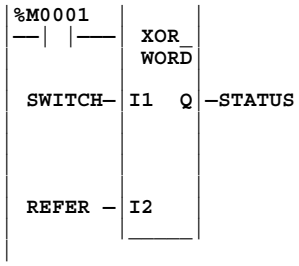
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		•	•	•	•	•	•	•	•	•	•	
I2		•	•	•	•	•	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example of an Alarm Circuit Using an XOR

In the following example, whenever enable contact %M0001 is on, the 16-bit string nicknamed SWITCH is compared to a reference bit string, nicknamed REFER. The SWITCH bit string is a group of bits that represent the on/off status of alarm switch contacts. The REFER bit string represents the normal or non-alarm status of these bits. If the state of any SWITCH bit is different from its corresponding REFER bit, their corresponding output at Q goes to logic 1. Under normal (no alarm) conditions, the value of the word nicknamed STATUS will be zero.



Bit Position	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
I1 (SWITCH)	0	1	0	1	1	1	1	0	1	1	0	0	0	0	0	0
I2 (REFER)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0
Q (STATUS)	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0

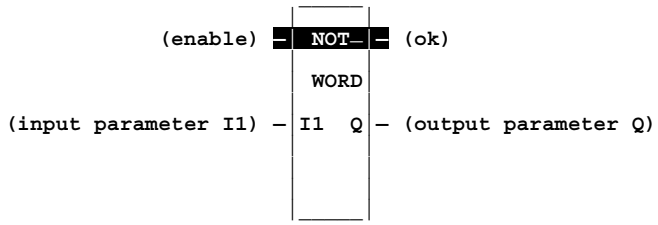
The data in STATUS could be used as an input to a Not Equal (NE) function, which would compare the word nicknamed STATUS to a constant of zero. If STATUS does not equal zero, the NE turns on its output, indicating the presence of an alarm.

The bits in STATUS that are equal to logic 1 can be identified with the BPOS (Bit Position) function, which would search the bits in STATUS and report the position (a number between 1 and 16) of the first bit (starting at bit 1) it encounters that is at logic 1. In the example above, the BPOS would output the number 4, indicating the fourth bit is a logic 1. To test for more than one bit, you could store a record of bit 4, use a BCLR (Bit Clear) function to clear bit 4, then repeat the BPOS test to find the next bit that is equal to logic 1 (bit 9 in the example above). This process can be repeated until no more non-zero bits are found. Note that the BCLR and BPOS functions are discussed in detail elsewhere in this chapter.

NOT (WORD)

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.

The NOT function executes and passes power flow for each scan that it's enable input is on.



Parameters

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains the 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.

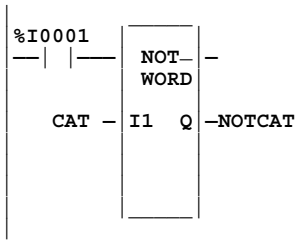
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		•	•	•	•	•	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

• Valid reference or place where power may flow through the function. † %SA, %SB, or %SC only; %S cannot be used.

Example

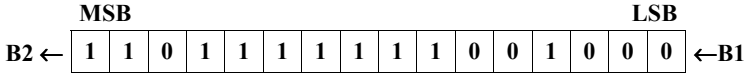
In the following example, whenever input %I0001 is set, the bit string represented by the nickname NOTCAT is set to the inverse of bit string CAT, as seen in the truth table below.



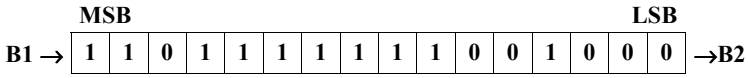
CAT	1	1	0	1	0	1	0	0	1	1	0	1	0	0	0	1
NOTCAT	0	0	1	0	1	0	1	1	0	0	1	0	1	1	1	0

SHL and SHR (WORD)

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.



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.



A string length of 1 to 256 words can be selected for either function.

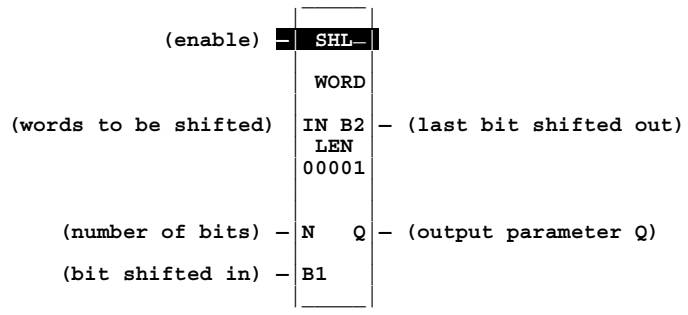
If the number of bits to be shifted (N) is greater than the number of bits in the array (LEN) * 16, or if the number of bits to be shifted is zero, then the array (Q) is filled with copies of the input bit (B1), and the input bit is copied to the B2 output. If the number of bits to be shifted is zero, then no shifting is performed; the input array is copied into the output array; and input bit B1 is copied to the B2 output.

The bits being shifted into the beginning of the string are specified via input parameter B1, which requires a contact to the power rail. 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) of B1. The B1 input can be controlled by

- An ALW_ON (%S07) contact, which holds B1 permanently at logic 1.
- An ALW_OFF (%S06) contact, which holds B1 permanently at logic 0.
- A contact from an internal coil such as %M or %Q that lets you change the value.
- A %I contact that lets you change the value from an input contact.

The SHL or SHR function passes power flow to the right, unless the number of bits specified to be shifted 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 SHL/SHR instructions execute each scan that their enable input is on. Output B2 holds the value of the last bit shifted out; for example, if four bits were shifted, B2 would contain be the value (either 1 or 0) of the fourth bit shifted out.



Parameters

Parameter	Description
enable	When enable is logic 1, the shift is performed.
IN	IN contains the address of the first word to be shifted.
N	N contains the number of places (bit positions) that the array is to be shifted.
B1	B1 contains the bit value (0 or 1) to be shifted into the array.
B2	B2 contains the bit value (0 or 1) of the last bit shifted out of the array.
Q	Output Q contains the first word of the shifted array.
LEN	LEN is the number of words (1 – 256) in the array to be shifted.

Valid Memory Types

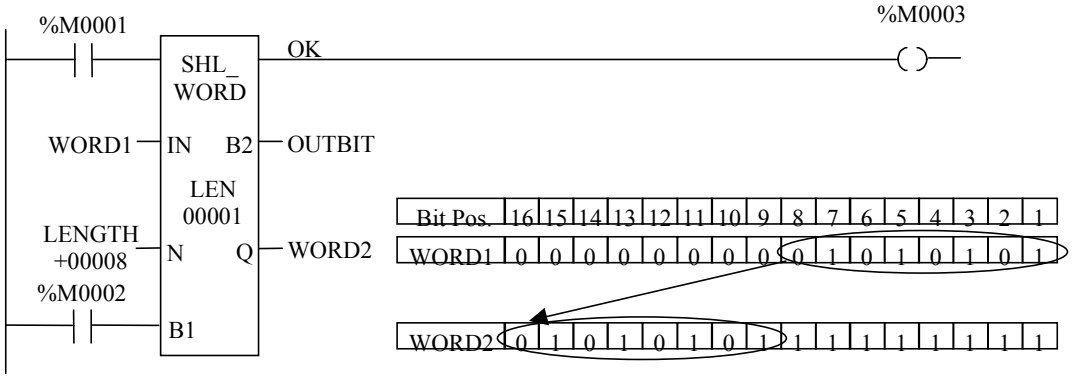
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
N		•	•	•	•		•	•	•	•	•	
B1	•											
B2	•											•
Q		•	•	•	•	•†	•	•	•	•		

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

† %SA, %SB, or %SC only; %S cannot be used.

Example

In the following example, when input %M0001 is on, the SHL makes a copy of the bit string at IN (nicknamed WORD1). Then, in the copy, it shifts all bits to the left by 8 bit positions (specified by the value at N). The bits from bit positions 9-16 are shifted out (discarded), and the bits that were in positions 1-8 now occupy bit positions 9-16. Bit positions 1-8, which were "vacated" when bits 1-8 were shifted, are filled with ones because, for this example, contact %M0002 is closed, making the B1 input equal to logic 1. Finally, the shifted/filled word is written to the address at output Q (nicknamed WORD2). The original WORD1 at IN is not changed. Output B2 equals zero since the last bit shifted out was logic zero (the bit that occupied bit position 9), and coil %M0003 is on because the function worked correctly and therefore produced power flow at its OK output.



ROL and ROR (WORD)

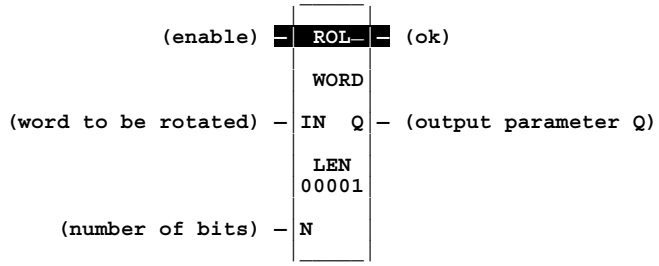
The Rotate Left (ROL) function rotates all the bits in a string a specified number of places to the left. When rotation occurs, the specified number of bits is rotated out of the input string to the left and back into the string on the right.

The Rotate Right (ROR) function rotates all bits in a string a specified number of places 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 can be selected for either function.

The number of places specified for rotation at input N 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 rotation 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 rotate function executes each scan that the enable input is on.



Parameters

Parameter	Description
enable	When the enable input is on, the rotation is performed.
IN	IN contains the address of the first word to be rotated.
N	N contains the number of places (bit positions) that the array is to be rotated.
ok	The ok output is energized when the rotation function is enabled and the rotation length (at N) is greater than zero but 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 (1 – 256) in the array to be rotated.

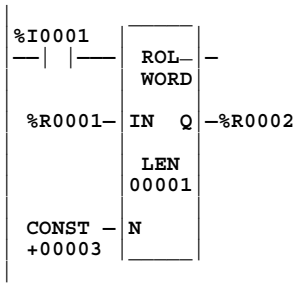
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
N		•	•	•	•		•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

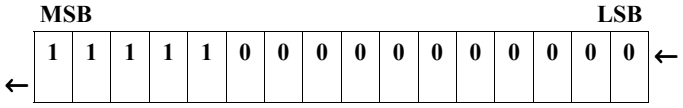
• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example

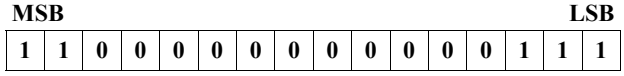
In the following example, whenever enable input %I0001 is on, the ROL makes a copy of the input string at IN. Then, in the copy, it rotates the input bit string 3 bits (specified by the value of input N) and places the result in %R0002. After execution of this function, the input bit string %R0001 is unchanged. However, if you wish to rotate the input string, use the same reference address for IN and Q.



%R0001:



%R0002 (after rotation occurs):



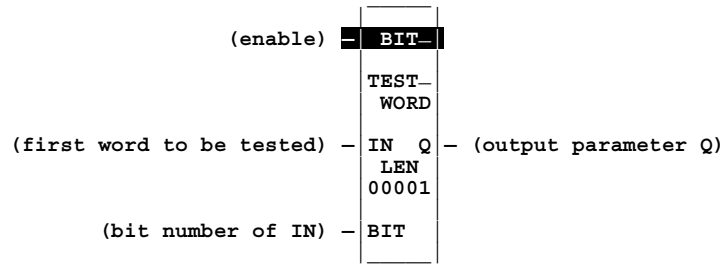
BTST (WORD)

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 specified by the following formula, then Q is set OFF.

Formula: 1 ≤ BIT ≤ (16 * LEN)

A string length of 1 to 256 words can be selected.



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 ≤ BIT ≤ (16 * LEN)).
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.

Note

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15.

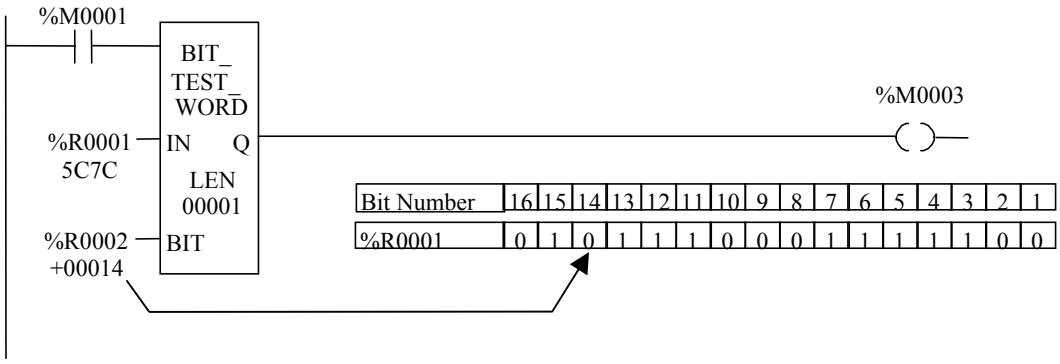
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
BIT		•	•	•	•		•	•	•	•	•	
Q	•											•

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

Example

In the following example, whenever enable input %M0001 is on, bit 14 in word %R0001 is tested (bit 14 is specified by the value in %R0002). Since bit 14 is zero in the value shown for %R0001 (5C7C), output Q does not turn on. Note that this function can only be a WORD type; therefore, any memory address used at IN will appear on a Logicmaster screen in hexadecimal format. However, the value at BIT will appear in integer format regardless of whether a constant or memory address is used.

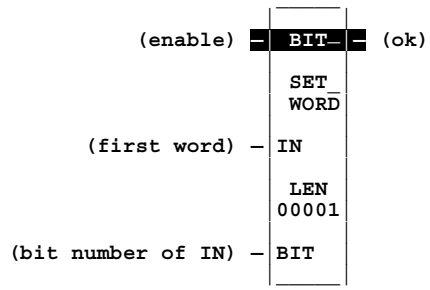


BSET and BCLR (WORD)

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 can be selected. The function passes power flow to the right, unless the value for BIT is outside the range ($1 \leq BIT \leq (16 * LEN)$). Then, ok is set OFF. For example, if LEN is set to 1, then the length of the bit string to be tested is 16. If, in this case, the number at BIT was 17 or higher, it would be out of range, so the ok output would not come on.



Parameters

Parameter	Description
enable	When the enable input is on, the bit operation is performed.
IN	IN contains the address of the first word of the bit string to be operated on.
BIT	BIT contains the bit number of IN that should be set or cleared. Valid range is ($1 \leq BIT \leq (16 * LEN)$).
ok	The ok output is energized whenever enable is energized, unless the value at the BIT input is outside the valid range.
LEN	LEN is the number of words in the bit string whose starting address is configured at IN..

Note

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15.

Valid Memory Types

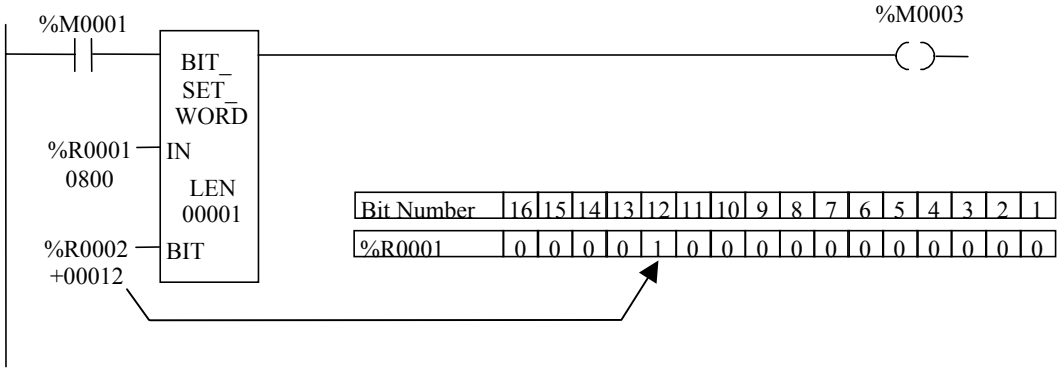
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	†	•	•	•	•		
BIT		•	•	•	•		•	•	•	•	•	
ok	•											•

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

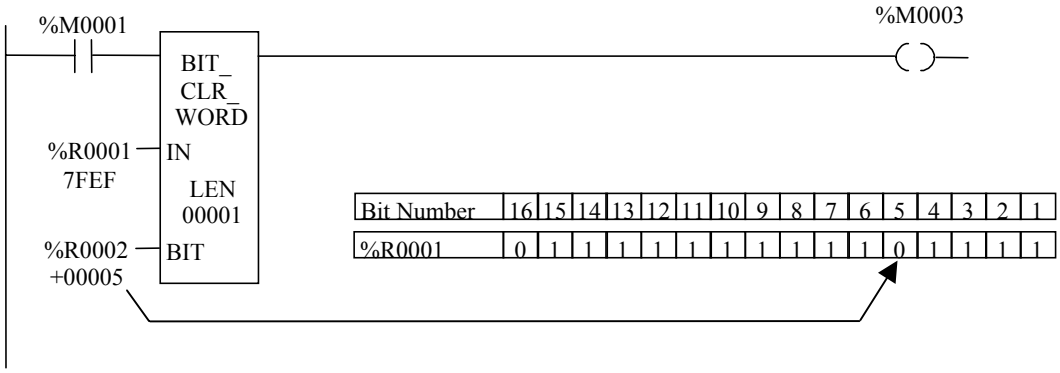
Examples

Note that the Bit Set and Bit Clear functions can only be WORD types; therefore, any memory address used at IN will appear on a Logicmaster screen in hexadecimal format. However, the value at BIT will appear in integer format whether a constant or memory address is used.

In the following example, when input %M0001 is on, bit 12 (specified by the BIT input) of the string beginning at reference %R0001 (the address at the IN input) is set to 1 (set).



In the next example, when input %M0001 is on, bit 5 (the value of the BIT input) of the string beginning at reference %R0001 (the address at the IN input) is set to 0 (cleared).



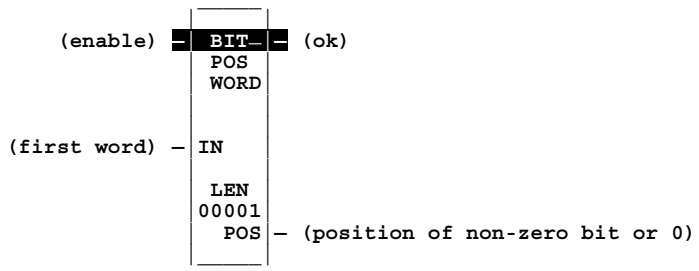
BPOS (WORD)

The Bit Position (BPOS) function is used to locate in a bit string, a bit whose value is logic 1.

Each sweep that the function is enabled, it 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.



Parameters

Parameter	Description
enable	When the enable input is on, a bit search operation is performed.
IN	IN contains the first word of the bit string to be operated on.
ok	The ok output is energized whenever enable is energized.
POS	The position of the first non-zero bit found, or zero if a non-zero bit is not found.
LEN	LEN is the number of words in the bit string.

Note

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15.

Valid Memory Types

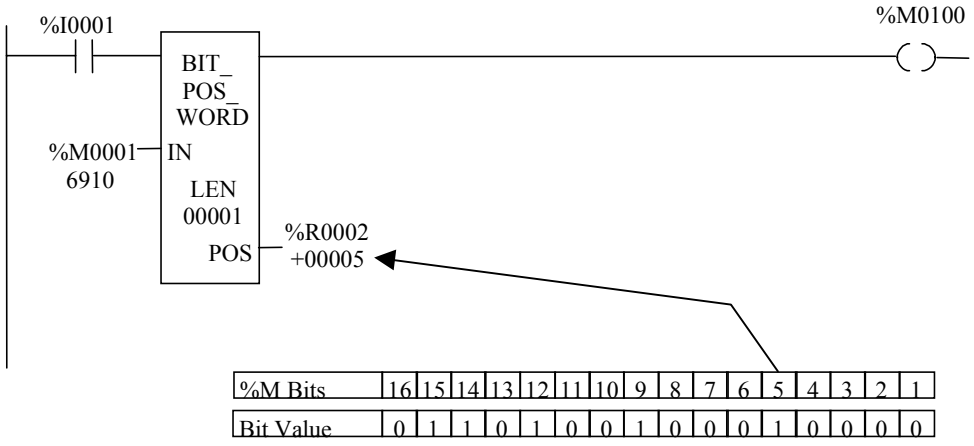
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
POS		•	•	•	•		•	•	•	•		
ok	•											•

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

Example

Note that the Bit Position function can only be a WORD type; therefore, any memory address used at IN will appear on a Logicmaster screen in hexadecimal format. However, the value at POS will appear in integer format. Logicmaster displays the first 16 bits at IN in hexadecimal format.

In the following example, if %I0001 is on, the bit string starting at %M0001 is searched until a bit equal to 1 is found, or until the entire bit string has been searched. Coil %M0100 is turned on. If a bit equal to 1 is found, its location within the bit string is written to %R0002; otherwise a value of 0 is written to %R0002. In the example shown, bit 5 is the first logic 1 encountered by the search (which starts at bit 1), so the value written to %R0002 is 5.



MSKCMP (WORD, DWORD)

The Masked Compare (MSKCMP) function (*available for Release 4.41 or later CPUs*) is used to compare the contents of two separate bit strings with the ability to mask selected bits. The length of the bit strings to be compared is specified by the LEN parameter (where the value of LEN specifies the number of 16-bit words for the MSKCMP word-type function or 32-bit words for the MSKCMP double-word type function).

When its enable input is on, the function compares the bits in the first string with the corresponding bits in the second string. Comparison continues until a miscompare is found, or until the end of the string is reached. The function executes each scan that the enable input is on, so, for many applications, a "one-shot" contact is used for the enable input.

The BIT input is used to store the bit number where the next comparison should start (where a *0* indicates the first bit in the string). The BN output is used to store the bit number where the last comparison occurred (where a *1* indicates the first bit in the string). Using the same reference for BIT and BN causes the compare to start at the next bit position after a miscompare; or, if all bits compared successfully upon the next invocation of the function block, the compare starts at the beginning.

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

If All Bits in I1 and I2 are the Same

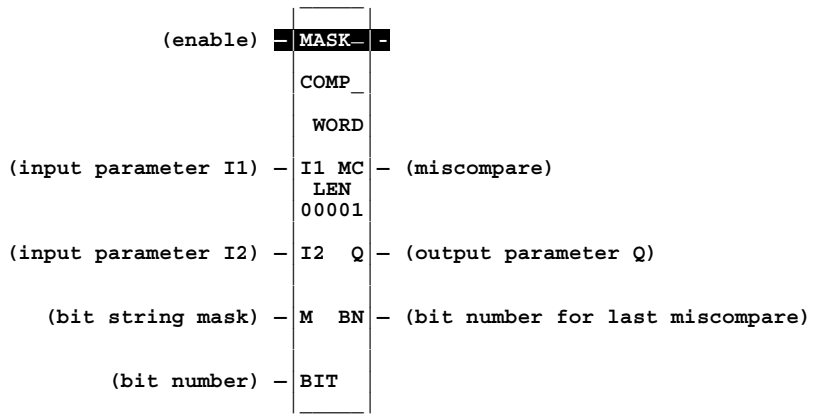
If all corresponding bits in strings I1 and I2 match, the function sets the "miscompare" output MC to 0 and BN to the highest bit number in the input strings. The comparison then stops. On the next invocation of MSKCMP, BN will be reset to 0.

If a Miscompare is Found

When the two bits currently being compared are not the same, the function checks the correspondingly numbered bit in string M (the mask). If the mask bit is a *1*, the miscompare is ignored and the comparison continues until it reaches another miscompare or the end of the input strings.

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

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



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 for the bit number where the next comparison should start.
MC	Goes to a logic 1 for one scan if a miscompare has occurred. A set coil can be used on this output if it is desired to "capture" the output beyond one scan.
Q	Output copy of the mask (M) bit string.
BN	Number of the bit where the last compare occurred.
LEN	LEN is the number of words in the bit string.

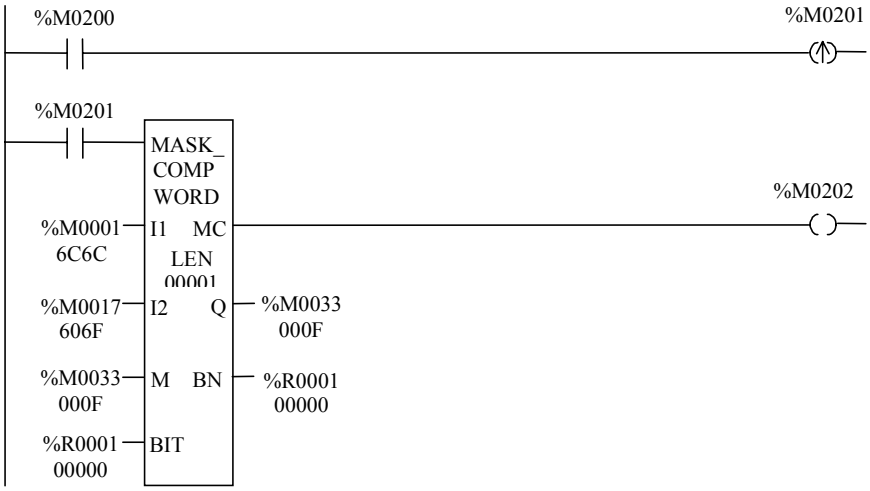
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	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		•	•	•	•	•	•	•	•	•	•	
LEN											•‡	
MC	•											•
Q		o	o	o	o	o†	o	•	•	•		
BN		•	•	•	•	•	•	•	•	•		

• 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.
‡ Max const value of 4095 for WORD and 2047 for DWORD.

Example 1 – MSKCMP Instruction

When %M0200 closes, the contact from the %M0201 transition coil closes for one scan, which enables the MSKCMP function to execute once. %M0001 through %M0016 (I1) are compared with %M0017 through %M0032 (I2). %M0033 through %M0048 (M) contains the mask value. The value in %R0001 (BIT) determines at which bit position (0) the comparison starts within the two input strings at I1 and I2.



Condition Before the First MSKCMP Execution

The contents of the input references before the MSKCMP executes are as follows:

%M Bits	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Input 1(I1)	0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0

%M Bits	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
Input 2(I2)	0	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

%M Bits	48	47	46	45	44	43	42	41	40	39	38	37	36	35	35	33
Mask (M/Q)	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

BIT/BN (%R0001) = 0
MC (%M0202) = OFF

Condition After the First MSKCMP Execution

The following table shows the contents of the Mask (M/Q) references after the MSKCMP executes one time. (I1 and I2 are still at the values shown above.) Since the ninth bit produced a miscompare, the ninth bit (%M0041) in the Mask string is set to logic 1, BIT/BN contains a value of 9, and the MC output turned on for one scan. Although the first and second bit positions are not equal, they do not produce a miscompare because the mask bits are 1 for these positions.

%M Bits	48	47	46	45	44	43	42	41	40	39	38	37	36	35	35	33
Mask (M/Q)	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1

BIT/BN (%R0001) = 9
MC (%M0202) = ON (for one scan)

Example 2 - Fault Detection with a Masked Compare Function

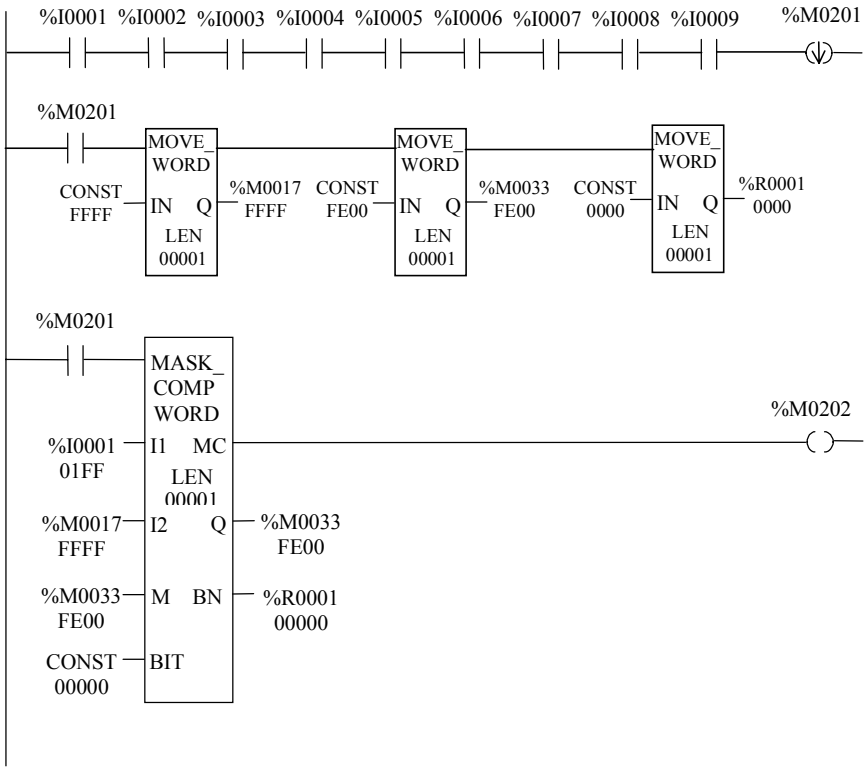
Intermittent problems can be difficult to troubleshoot. One example is when several switches are arranged in a series circuit that energizes a fault relay. Under normal conditions, all switches are closed and the fault relay is energized (a “fail-safe” arrangement). When a fault occurs, one of the contacts opens and the fault relay drops out. If the faulted contact remains open, a troubleshooter will be able to easily determine which switch caused the fault. However, sometimes a contact only opens for a brief time, perhaps for less than a second, then closes again. This causes the fault relay to drop out briefly and shut down the process. Since the contact closes again, everything appears normal.

To help with such a problem, the following circuit acts as a “fault catcher” in that it detects which contact opened and stores its number in a register. In the first rung, contacts from the input switches, which are each wired to an input module point (%I1 – %I9), are programmed in series to energize %M0021, a negative transition coil.

The second rung initializes the MSKCMP so it is ready to capture the fault. The first Move instruction writes all logic ones to the I2 input of the MSKCMP. The second Move writes values of 1 to bits 10—16 of the mask word (so that these bits are ignored), since only the first nine bits of the compared words (the MSKCMP uses full words) are needed for switches %I0001—%I0009. The third Move zeroes the output register, %R0001, so it is ready to report the latest fault.

During normal operation, the first nine bits on input I1 of the MSKCMP are at logic 1 since the switches are all closed. Input I2 is initialized with all logic 1s since that is the normal condition to which the input switches are compared. The mask has 1s in bits 10—16 because these bits are not used since there are nine input switches. When a switch opens, %M0201’s contacts close for one scan. This causes the initializing moves to occur in the second rung, and in the third rung, the MSKCMP is enabled. The MSKCMP compares the input switches against the logic 1s at its I2 input, identifies which switch is logic 0 (open), and writes the bit number of the open switch to the BN (%R0001) output. The bits are numbered from 1—9 beginning with %I1. For example, if %I4 were to open, %R0001 would contain the number 4.

Note that, in this circuit, if a switch opens and closes again, coil %M0201 drops out and picks back up, but the number of the switch that opened will be stored in %R0001. However, if a switch opens again, for example, the machine operator pushes an emergency stop button or opens a safety gate, the masked compare activates again and writes the number of the latest switch opening in %R0001. This means that the equipment should be left untouched after the fault occurs until the value in %R0001 can be checked. If this is not practical, an additional Move instruction could be used.


%I0001 %I0002 %I0003 %I0004 %I0005 %I0006 %I0007 %I0008 %I0009
%M0201
%M0201
MOVE_ WORD
CONST FFFF
IN Q
LEN 00001
%M0017 FFFF
MOVE_ WORD
CONST FE00
IN Q
LEN 00001
%M0033 FE00
MOVE_ WORD
CONST 0000
IN Q
LEN 00001
%R0001 0000
%M0201
MASK_ COMP WORD
%I0001 01FF
I1 MC
LEN 00001
%M0017 FFFF
I2 Q
%M0033 FE00
%M0033 FE00
M BN
%R0001 00000
CONST 00000
BIT
%M0202

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 256 words, except MOVE_BIT 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.	9-5
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, %AI, or %AQ) memory. The maximum length allowed is 256 words.	9-7
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
COMMREQ	Communications Request	Allow the program to communicate with an intelligent module, such as a Genius Communications Module or a Programmable Coprocessor Module.	9-15

MOVE (BIT, INT, WORD, 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 is enabled, it copies data from input parameter IN to 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 input and output references.

For the BIT type there is another consideration. If a BIT array specified on the Q parameter does not encompass all of the bits in a byte, the transition bits associated with that byte (which are not in the array) will be cleared when the MOVE_BIT receives power flow.

Input IN can be either a reference for the data to be moved or a constant. If a constant is specified, then the constant value is placed in the location specified by the output reference. For example, if a constant value of 4 is specified for IN, and the length (LEN) equals 1, then 4 is placed in the memory location specified by Q. If the length is greater than 1 and a constant is specified, then the constant is placed in the memory location specified by Q and the locations following, up to the length specified. For example, if the constant value 9 is specified for IN and the length equals 4, then 9 is placed in the memory location specified by Q and also in the three locations following.

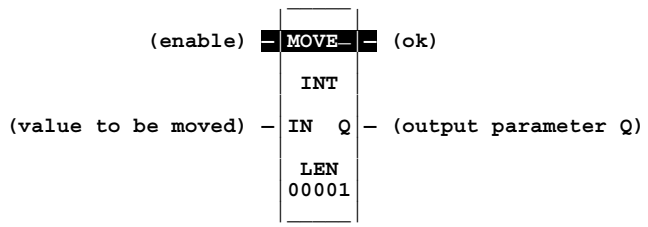
The LEN operand specifies the number of:

- Words to be moved for MOVE_INT and MOVE_WORD.
- Bits to be moved for MOVE_BIT.
- Real numbers to be moved for MOVE_REAL.

Note

The REAL data type is only available on 35x and 36x series CPUs, Release 9 or later, and all releases of CPU352 and 37x.

The function passes power to the right whenever power is received.



Parameters

Parameter	Description
enable	When the function is enabled, the move is performed.
IN	IN contains the value to be copied (moved). For MOVE_BIT, any discrete reference may be used; it does not need to be byte aligned. However, a 16-bit value, beginning with the reference address specified, is displayed on the Logicmaster screen.
ok	The ok output is energized whenever the function is enabled.
Q	When the move is performed, the value at IN is copied to Q. For MOVE_BIT, any discrete reference may be used; it does not need to be byte aligned. However, a 16-bit value, beginning with the reference address specified, is displayed on the Logicmaster screen.
LEN	LEN specifies the number of words or bits to be moved. For MOVE_WORD and MOVE_INT, LEN must be between 1 and 256 words. For MOVE_BIT, when IN is a constant, LEN must be between 1 and 16 bits; otherwise, LEN must be between 1 and 256.

Note

On 351, 352, 36x and 37x series CPUs, the MOVE_INT and MOVE_WORD functions do not support overlapping of IN and Q parameters, where the IN reference is less than the Q reference. For example, with the following values: IN=%R0001, Q=%R0004, LEN=5 (words), the %R0007 and %R0008 contents will be indeterminate; however, using the following values: Q=%R0001, IN=%R0004, LEN=5 (words) will yield valid contents.

Also, please note that only 35x and 36x series CPUs (Release 9.00 and later), and all releases of CPU35 and 37x have Floating Point capabilities and are therefore the only Series 90-30 CPUs capable of MOVE_REAL.

Valid Memory Types

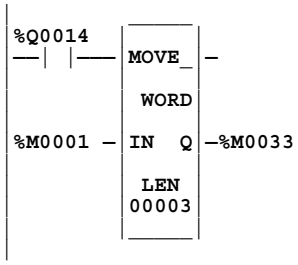
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	o	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	o†	•	•	•	•		

Note: For REAL data, the only valid types are %R, %AI, and %AQ.

- • Valid reference for BIT, INT, or WORD data, or place where power may flow through the function. For MOVE_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.
- o Valid reference for BIT or WORD data only; not valid for INT.
- † %SA, %SB, %SC only; %S cannot be used.

Example 1 - Overlapping Addresses (only for CPUs 311-341)

When enable input contact %Q0014 is ON, 48 bits are moved from memory location %M0001 to memory location %M0033. Even though the destination overlaps the source for 16 bits, the move is done correctly (except for the 35x and 35x CPUs as noted previously).



Before using the Move function:

INPUT (%M0001 through %M0048)

																1
%M0016	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
%M0032	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
%M0048	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

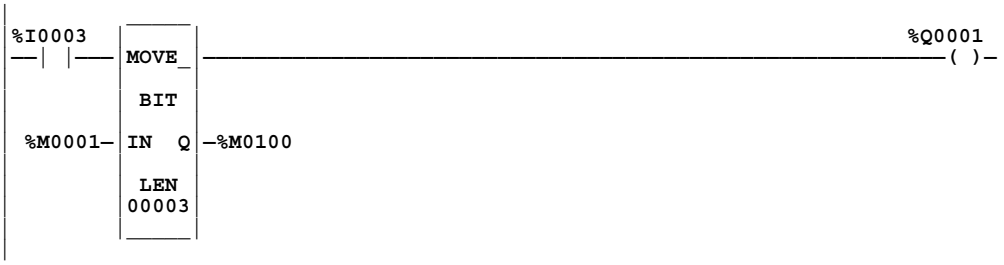
After using the Move function:

OUTPUT (%M0033 through %M0080)

																33
%M0048	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
%M0064	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
%M0080	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Example 2 – for all CPUs

In this example, whenever %I0003 is on, the values in the three bits %M0001, %M0002, and %M0003 are moved to %M0100, %M0101, and %M0102, respectively, and coil %Q0001 is turned on.



BLKMOV (INT, WORD, REAL)

Use the Block Move (BLKMOV) function to copy a block of seven constants to a specified location.

Note

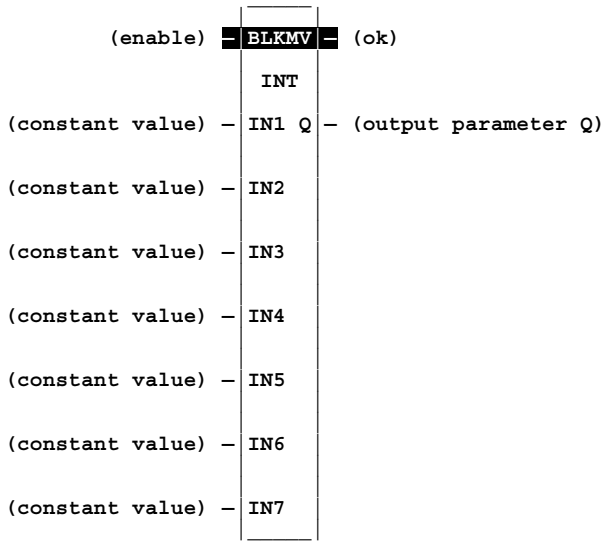
The REAL data type is only available on 35x and 36x series CPUs, Release 9 or later, and all releases of CPU352 and 37x.

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 — IN7 are displayed as signed decimals. For BLKMOV_WORD, IN1 — IN7 are displayed in hexadecimal. For BLKMOV_REAL, IN1— IN7 are displayed in Real format.

The function passes power to the right whenever it is enabled.



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	%R	%AI	%AQ	const	none
enable	•											
IN1 — IN7											•	
ok	•											•
Q		•	•	•	•	o†	•	•	•	•		

Note: For REAL data, the only valid types are %R, %AI, and %AQ.

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

o Valid reference for WORD data only; not valid for INT or REAL.

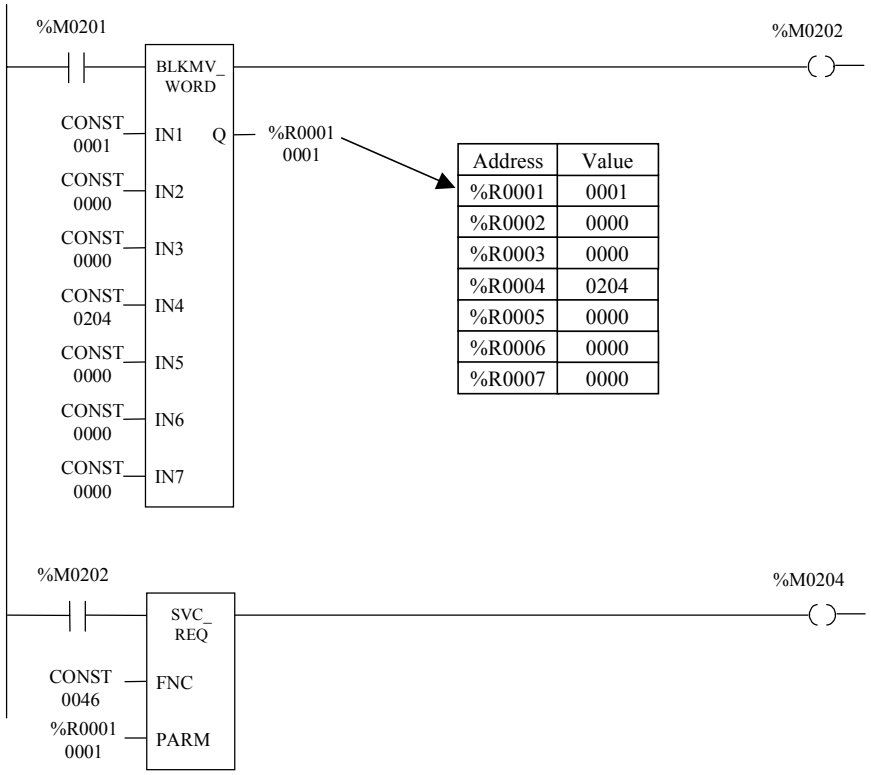
† %SA, %SB, %SC only; %S cannot be used.

Note

Floating Point capabilities exist only on 35x and 36x series CPUs, Release 9 or later, and all releases of CPU352 and 37x. These 90-30 CPUs are the only ones capable of BLKMV_REAL.

Example

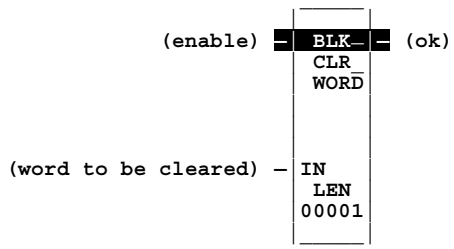
In the following example, when input enable contact %M0201 is on, the BLKMOV function copies the seven input constants into memory locations %R0001 (specified at output Q) through %R0007. If the BLKMV executes successfully, it turns on its OK output, which energizes %M0202. In turn, an %M0202 contact enables the Service Request function in the next rung, which uses %R0001 through %R0007 as its parameter block. (See Chapter 12 for more information on Service Request instructions.)



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. The function passes power to the right.



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

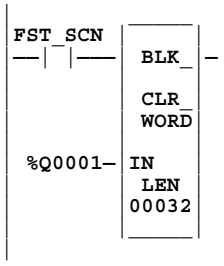
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•†	•	•	•	•		
ok	•											•

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

† %SA, %SB, %SC only; %S cannot be used.

Example

In the following example, at power-up, 32 words of %Q memory (512 points) beginning at %Q0001 are filled with zeros.



SHFR (BIT, WORD)

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

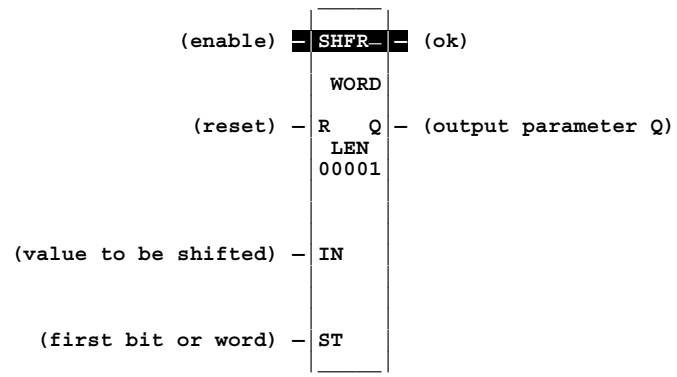
When assigning reference addresses, overlapping input and output reference address ranges in multi-word functions may produce unexpected results.

The SHFR function has four 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.

If the function receives power flow and reset is not active, each bit or word of the shift register is moved to the next highest reference. The last element in the shift register is shifted into Q. If Q has a unique address, the data shifted out of Q is discarded. However, if IN and Q are given the same address, the data will re-circulate in the shift register. The highest reference of the shift register element of IN is shifted into the vacated element starting at ST. The contents of the shift register are accessible throughout the logic program because they are all contained in addressable memory.

The function passes power to the right whenever power is received through the enable logic.

The function will execute once each scan while it is enabled; so it may be beneficial to use a "one-shot" type enable contact from a transition coil if it is desired to just shift one time for a given contact closure.



Parameters

Parameter	Description
enable	When the enable input is on and the R input is off, the shift is performed. Note that the SHFR will execute once for each scan that it is enabled.
R	When the R input is on, the shift register located at ST is filled with zeros.
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, 16 bits, starting with the reference address specified, are 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, starting with the reference address specified, are displayed online.
ok	The ok output is energized whenever the enable input is on and the R input is off.
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, 16 bits, starting with the reference address specified, are displayed online.
LEN	LEN determines the length of the shift register. For SHFR_WORD, LEN must be between 1 and 256 words. For SHFR_BIT, LEN must be between 1 and 256 bits.

Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
R	•											
IN		•	•	•	•	•	•	•	•	•	•	
ST		•	•	•	•	•†	•	•	•	•		
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

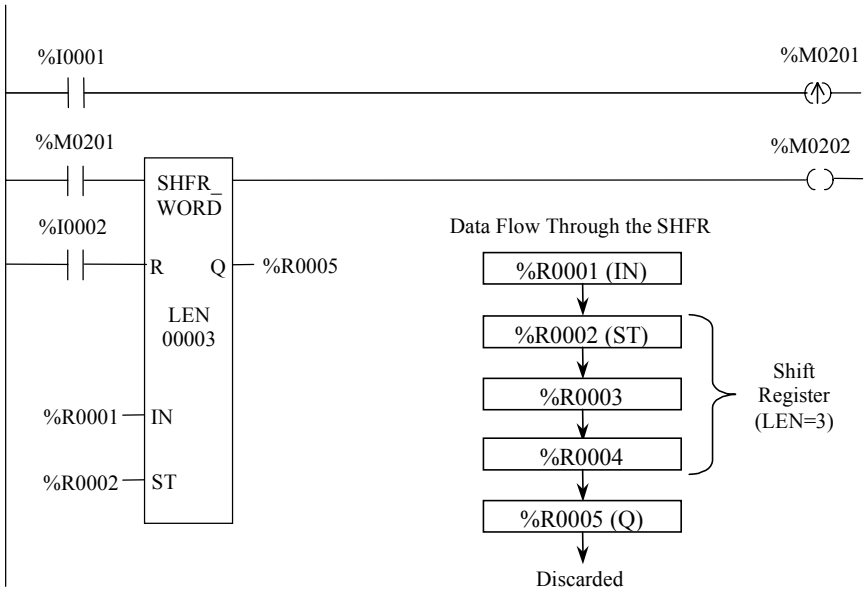
• Valid reference for BIT or WORD data, or place where power may flow through the function.
For SHFR_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.

† %SA, %SB, %SC only; %S cannot be used.

Example 1

In this example, the shift register operates on three (LEN=3) memory locations, %R0002 through %R0004. When the reset contact %I0002 is on, the three shift register words are set to zero.

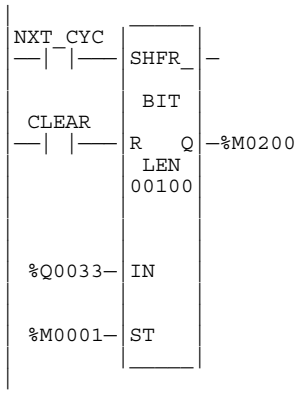
When contact %I0001 closes, the %M0201 contact at the SHFR's enable input closes for one scan. This shifts the data in %R0004 into output Q's address, %R0005 (the data that was in %R0005 is discarded). The data in %R0003 shifts into %R0004; the data in %R0002 shifts into %R0003, and the data in %R0001 (IN) shifts into %R0002 (ST). This data flow is shown in the figure below. If desired, data can be re-circulated by using the same address at IN and Q.



Example 2

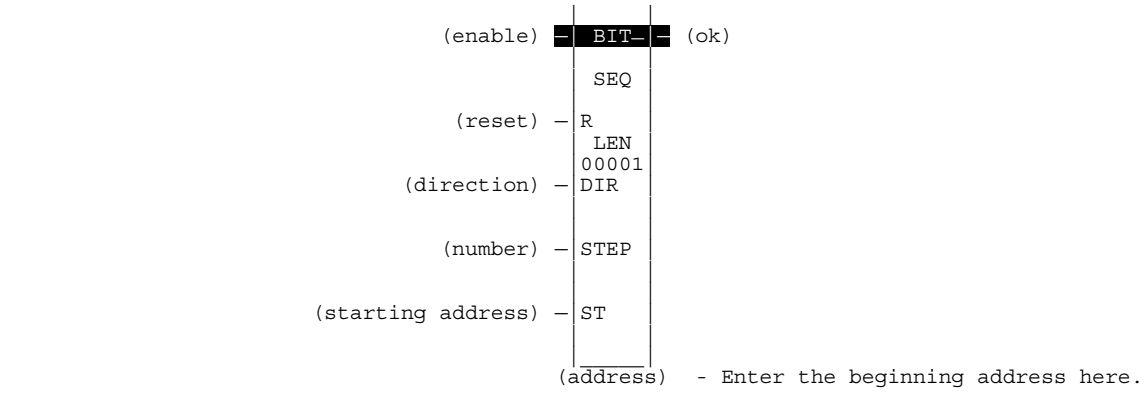
In Example 2, the shift register is a BIT type. With a LEN of 100, it operates on memory locations %M0001 through %M0100. When the reset reference CLEAR is active, the SHFR function fills %M0001 through %M0100 with zeros.

When NXT_CYC (a "one-shot" contact from a transition coil) is on and CLEAR is off, the SHFR function shifts the data in %M0001 through %M0100 up one bit. The bit in %Q0033 is shifted into %M0001 while the bit shifted out of %M0100 is written to Q (%M0200). The previous value of Q is discarded.



BITSEQ (BIT)

The Bit Sequencer (BITSEQ) function shifts a single logic 1 bit sequentially in a circular path through an array of bits. When the bit is shifted to the end of the array, it will wrap around to the other end of the array on the next shift and continue from there. The BITSEQ function has five input parameters and one output parameter.



Enable Input Requirement

The Bit Sequencer's Enable input requires a transition from logic zero to logic one in order for the function to execute one shift, and it will not execute again until it receives another positive-going Enable input transition. Therefore, using the contact from a positive transition coil for the Enable input is unnecessary.

R (Reset) Input

When this input is on, the Bit Sequencer will not execute.

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 specified in the STEP number parameter and all other bits are set to 0. If no STEP number is specified (STEP=0), the step is set to bit 1 and all other bits are set to 0.

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 DIR (direction) parameter. Then, the bit pointed to by the new step number is set to 1.

STEP Input

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

The parameter ST is optional. If it is not used (it is left equal to its default of zero), 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.

DIR (Direction) Input

The direction of bit rotation can be changed by turning the DIR input on or off. If on, the bit is incremented through the array. If off, the bit is decremented.

ST (Starting Address) Input and LEN (Length) Parameter

The ST input contains a memory location for the starting address of the sequencer array. The length of the array, in bits, is set by the LEN parameter. For example, if ST is %M0001 and LEN equals 16, the array is composed of %M0001 through %M0016. If ST is a %R address, then LEN determines how many consecutive bits in %R memory are included in the array. For example, if ST is %R0004, and LEN equals eight, only the first eight bits of register %R will be used in the array; the last eight bits of %R0004 will be ignored by the Bit Sequencer.

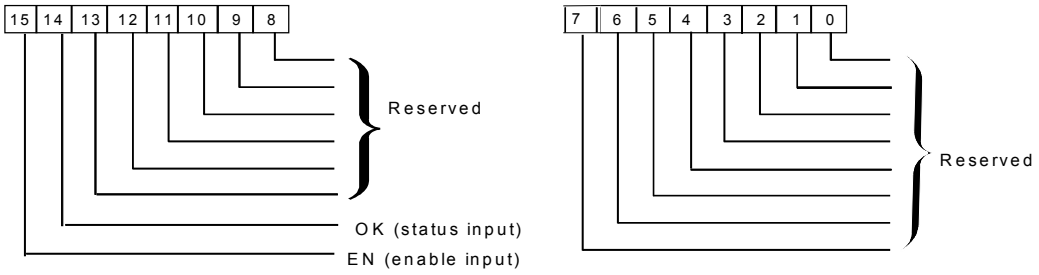
Control Block Memory Required for a Bit Sequencer

Each bit sequencer uses three words (registers) of %R memory to store the following information:

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

When you program a bit sequencer with Logicmaster, you must enter a beginning address for these three words (registers) directly below the graphic representing the function (see example on next page).

The control word stores the state of the Boolean inputs and outputs of its associated function block, as shown in the following format:



Note

Bits 0 through 13 are not used in the Control Block. Also, note that bits need to be entered as 1 through 16, ***NOT*** 0 through 15 in the STEP parameter.

Parameters

Parameter	Description
address	Address is the location of the bit sequencer's current step, length, and the last enable and ok statuses.
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 bits.

Note

Coil checking for the BITSEQ function checks 16 bits from the ST parameter, even when LEN is less than 16.

Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
DIR	•											
STEP		•	•	•	•		•	•	•	•	•	•
ST		•	•	•	•	•†	•	•	•	•		•
ok	•											•

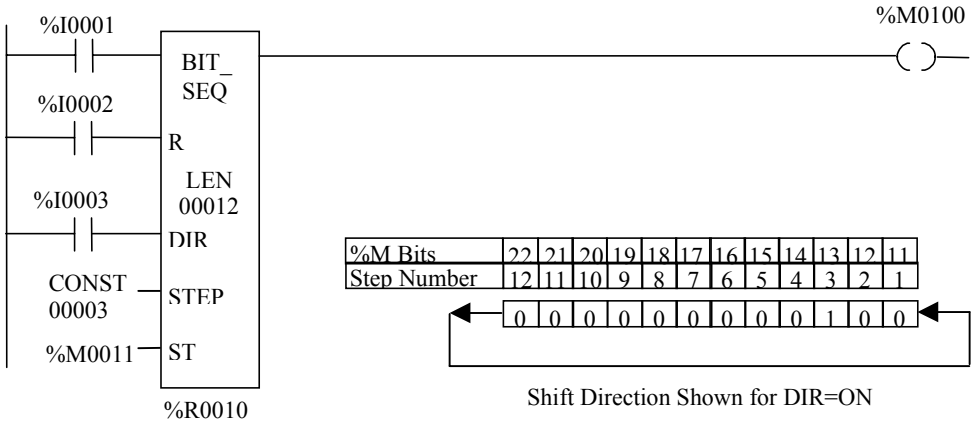
• Valid reference or place where power may flow through the function.

† SA, %SB, %SC only; %S cannot be used

Example

In the following example, the Bit Sequencer operates on bits %M0011 (specified in the ST input) through %M0022 (since LEN equals twelve). Its three-word control block is stored in registers %R0010, %R0011, and %R0012. When %I0002 (on the R input) is on, the sequencer is reset, which means that the bit for step three (specified in the STEP input) will be set to logic one and all other bits will be set to zero.

When %I0001 goes to logic 1 (with %I0002 off), the bit for step number 3 is cleared and either the bit for step number 4 will be set if DIR is on, or the bit for step number 2 will be set if DIR is off.



COMMREQ

Use the Communication Request (COMMREQ) function if the program needs to communicate with an intelligent module, such as a Genius Communications Module or a Programmable Coprocessor Module.

Note

The information presented on the following pages shows the general format of the COMMREQ function. You will need additional information to program the COMMREQ for each type of device. Programming requirements for each module that uses the COMMREQ function are described in the module's documentation.

The COMMREQ function has three input parameters and one output parameter. When the COMMREQ function receives power flow, a command block of data is sent to the intelligent module. The command block begins at the reference specified using the parameter IN. The rack and slot # of the intelligent module are specified in 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 within that time, program execution resumes. This is referred to as **`WAIT`** mode.

The Function Faulted (FT) output may be set ON if:

1. The specified target (SYSID) is not present in that location.
2. The specified task (TASK) number is not valid for the targeted device
3. The data length is 0 (in the Command Block).
4. The device's status pointer address (part of the Command Block) does not exist. This may be due to an incorrect memory type selection, or an address within that memory type that is out of range.

Command Block

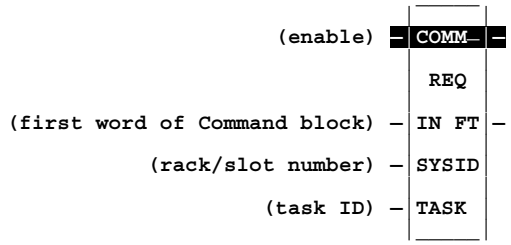
The Command Block provides information to the targeted intelligent module. It contains the command number to be performed as well as any data to be transferred.

The address of the Command Block is specified at the IN input to the COMMREQ function. This address may be in any word-oriented area of memory (%R, %AI, or %AQ). The length of the command block depends on the type of module targeted by the COMMREQ and the amount of data to be sent.

The command block has the following structure:

Length (in words)	address
Wait/No Wait Flag	address + 1
Status Pointer Memory	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 command block memory using an appropriate programming function such as a Block Move or a series of Moves.



Parameters

Parameter	Description
enable	While the enable input is on, the communications request is performed once per scan. If it is not desirable to send the COMMREQ multiple times, the enable input should be a contact from a Transition Coil.
IN	IN contains the starting address of the first word of the command block.
SYSID	SYSID contains the rack number (most significant byte) and slot number (least significant byte) of the targeted module.
TASK	TASK contains the task ID of the process on the targeted module.
FT	The FT (fault) output is energized if an error is detected processing the COMMREQ.

Note

The Series 90-30 COMMREQ does **not** have an OK output.

Valid Memory Types

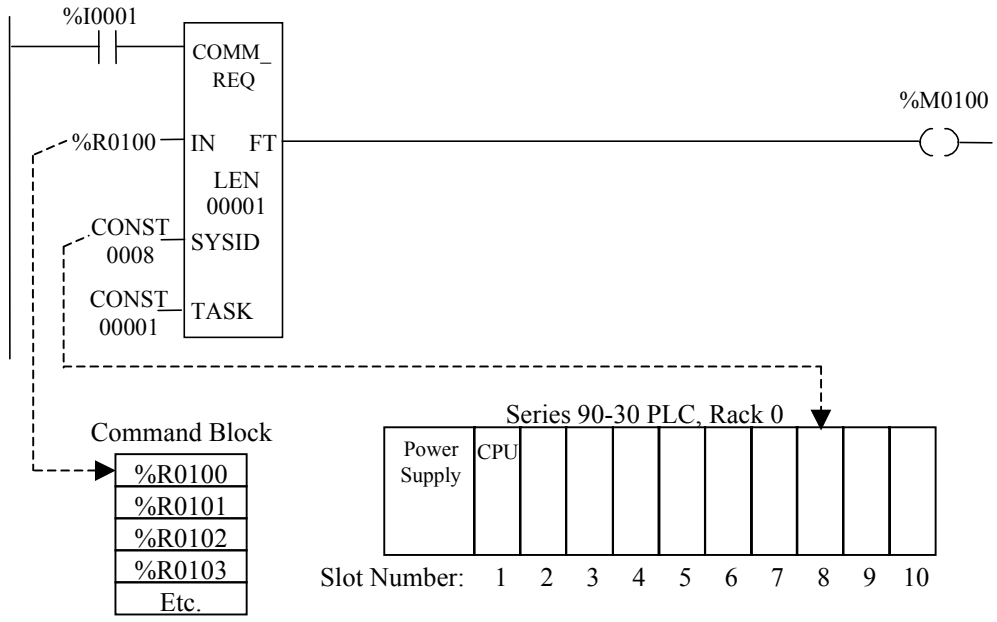
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN								•	•	•		
SYSID		•	•	•	•		•	•	•	•	•	
TASK								•	•	•	•	
FT	•											•

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

Example

In the following example, when enable input %I0001 is on, a command block starting at %R0100 (specified at the IN input) is sent to communications task 1 (TASK input = 1) in the module located at rack 0, slot 8 (SYSID=0008) of the PLC. If an error occurs while processing the COMMREQ, the Fault (FT) output turns on, which turns on %M0100.

Notice that the address at input IN specifies the starting address of the Command Block. Also, the hex. number at SYSID specifies the rack and slot number of the targeted module; the high byte refers to the rack number and the low byte refers to the slot number. Therefore, the SYSID of 0008 in the example refers to rack 00 and slot 08, as shown. Rack 0 (zero) always refers to the main or CPU rack, so if the targeted module was in an expansion or remote rack, the high byte of SYSID would contain a non-zero number that corresponds to the configured rack number where the targeted module is located.



Chapter 10 Table Functions

Table instructions are used to perform the following functions:

Abbreviation	Function	Description	Page
ARRAY_MOVE	Array Move	Copy a specified number of data elements from a source array to a destination array.	10-2
SRCH_EQ	Search Equal	Search for all array values equal to a specified value.	10-7
SRCH_NE	Search Not Equal	Search for all array values not equal to a specified value.	10-7
SRCH_GT	Search Greater Than	Search for all array values greater than a specified value.	10-7
SRCH_GE	Search Greater Than or Equal	Search for all array values greater than or equal to a specified value.	10-7
SRCH_LT	Search Less Than	Search for all array values less than a specified value.	10-7
SRCH_LE	Search Less Than or Equal	Search for all array values less than or equal to a specified value.	10-7

The maximum length allowed for these functions is 32,767 bytes or words, or 262,136 bits (bits are available for ARRAY_MOVE only).

Table functions operate on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
BIT *	Bit data type.
BYTE	Byte data type.
WORD	Word data type.

* Only available for ARRAY_MOVE.

The default data type is signed integer. The data type can be changed after selecting the specific data table function in the ladder logic software. To compare data of other types or of two different types, first use the appropriate conversion function (described in chapter 11, “Conversion Functions”) to change the data to one of the data types listed above.

ARRAY_MOVE (INT, DINT, BIT, BYTE, WORD)

Arrays and Data Elements Defined

For the purpose of this discussion, an **array** is a grouping of contiguous addressable PLC memory, such as %R0100 through %R0120. A **data element** is the data held in one unit of array memory. For example, if an array is a Bit type, then each data element is held in a single bit of memory, such as %M0001 (or it could be a single bit in register-type memory). Or, if an array is a Word type, then each data element is held in a 16-bit word of memory, such as %R0100 (or it could be 16 consecutive %I bits). See the "Valid Memory Types" table for more information on this.

Index Numbers

Each data element of an array has a reference number called an **index** number, which is automatically assigned by the PLC. The index number indicates the data element's position in the array. The data elements are numbered in ascending order, starting with the lowest memory address in the array, which is assigned index number one.

For example, the following Word-type array has a starting address of %R0105. It has ten data elements, whose index numbers are 1 through 10.

Address	Index No.
%R0105	1
%R0106	2
%R0107	3
%R0108	4
%R0109	5
%R0110	6
%R0111	7
%R0112	8
%R0113	9
%R0114	10

The Array Move Instruction

Use the Array Move function to copy a specified number of data elements from a source array to a destination array. Each array referenced by an Array Move instruction has an equal number of data elements. The Array Move allows the relative locations involved in the move to be different between the source and destination arrays. For example, three data elements, starting at index 5 in the source array, may be copied to three data elements in the destination array starting at index 7.

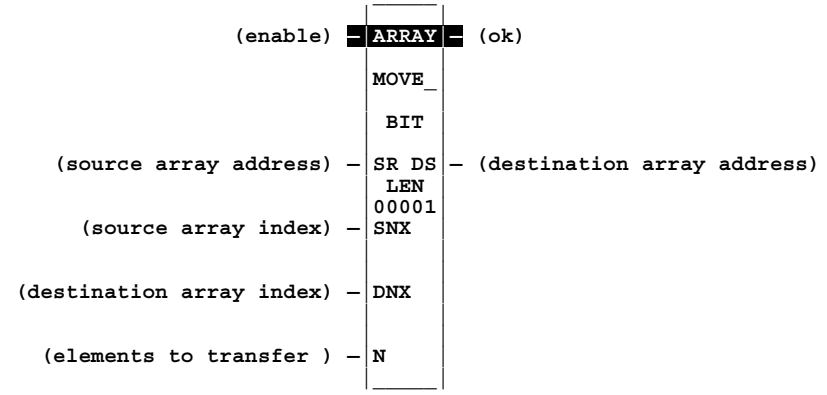
The ARRAY_MOVE function has five input parameters and two output parameters. When the function is enabled, the number of data elements in the count indicator (N) are copied from the input array starting with the indexed location specified at the SNX input. The data elements are written to the output array starting with the indexed location specified at DNX 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 on the Logicmaster screen 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 – 1) is greater than LEN. This formula is used by the PLC to ensure that no element outside the source array is referenced.
- (N + DNX – 1) is greater than LEN. This formula is used by the PLC to ensure that no element outside the destination array is referenced.
- SNX or DXN = 0.



Parameters

Parameter	Description
enable	When the enable input is on, the Array Move 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, 16 bits, beginning with the reference address specified, are displayed on the Logicmaster screen.
SNX	SNX contains the index number in the source array of the first data element to be copied.
DNX	DNX contains the index number in the destination array of the first element to be copied to.
N	The number of data elements to be copied.
ok	The ok output is energized whenever enable is 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, 16 bits, beginning with the reference address specified, are displayed online.
LEN	LEN specifies the number of data elements, starting at SR and DS, that make up each array.

Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
SR		o	o	o	o	Δ†	o	•	•	•		
SNX		•	•	•	•		•	•	•	•	•	
DNX		•	•	•	•		•	•	•	•	•	
N		•	•	•	•		•	•	•	•	•	
ok	•											•
DS		o	o	o	o	†	o	•	•	•		

• Valid reference or place where power may flow through the function.
For ARRAY_MOVE_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.

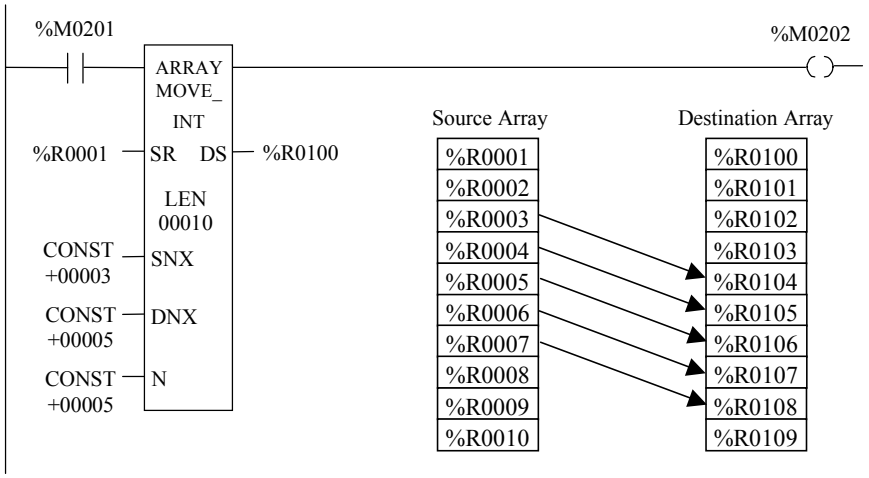
o Valid reference for INT, BIT, BYTE, or WORD data only; not valid for DINT.

Δ Valid data type for BIT, BYTE, or WORD data only; not valid for INT or DINT.

† %SA, %SB, %SC only; %S cannot be used.

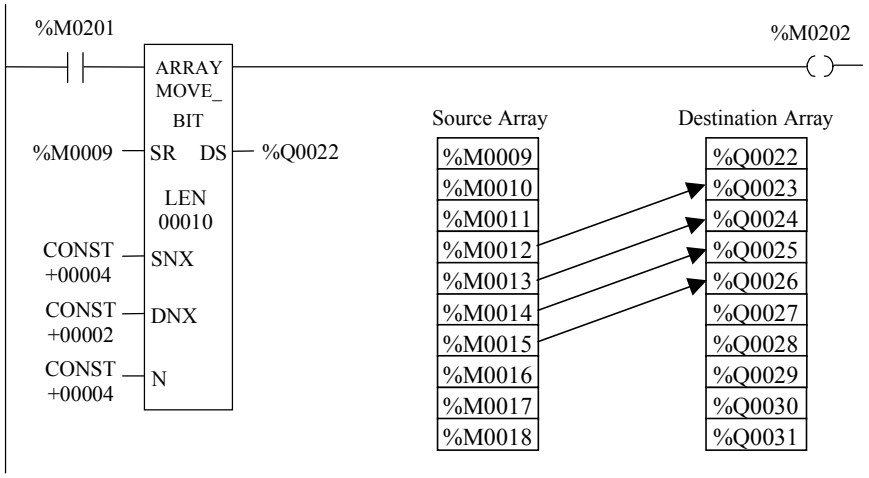
Example 1

In this example, both arrays are INT types that are10 elements (integers) long, specified by LEN=10. Their starting addresses are specified at SR and DS. When enable contact %M0201 is on, five data elements (specified by N=5) are copied from the source array to the destination array. The five copied data elements of the source array start with index number 3, since SNX=3. The locations copied to in the destination array start with index number 5, since DNX=5. So %R0003 through %R0007 of the source array are read and then copied into %R0104 through %R0108 of the destination array.



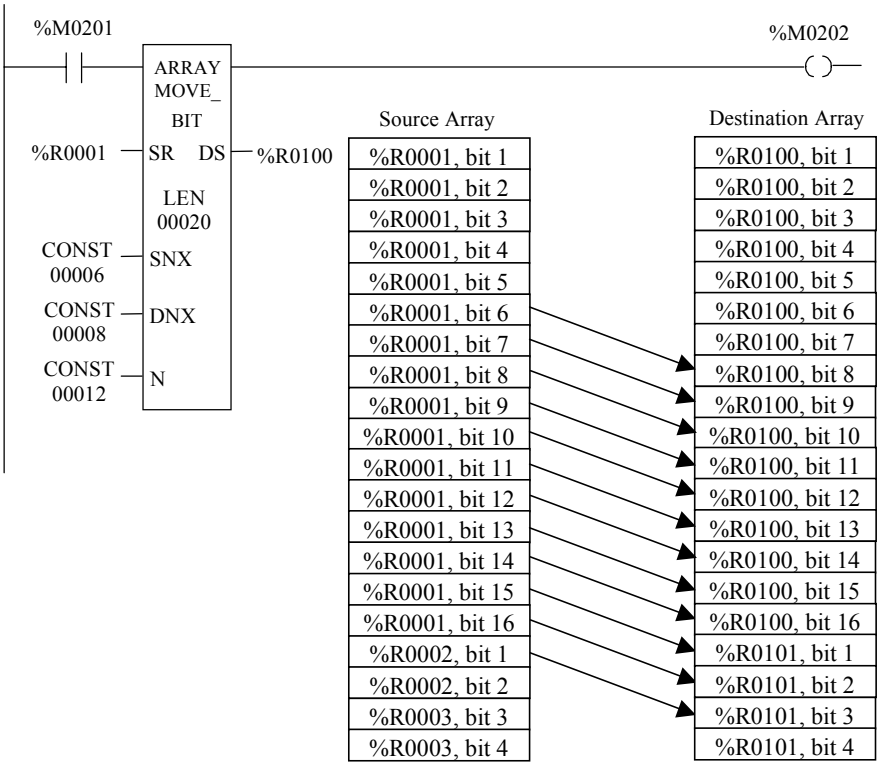
Example 2

In this example, both arrays are BIT types that are10 elements (bits) long, specified by LEN=10. Their starting addresses are specified at SR and DS. When enable contact %M0201 is on, four data elements (specified by N=4) are copied from the source array to the destination array. The four copied data elements of the source array start with index number 4, since SNX=4. The locations copied to in the destination array start with index number 2, since DNX=2. So %M0012 through %M0015 of the source array are read and then copied into %Q0023 through %Q0026 of the destination array.



Example 3

In this example, both arrays are BIT types that are 20 elements (bits) long, specified by LEN=20. Their starting addresses are specified at SR and DS. When enable contact %M0201 is on, 12 data elements (specified by N=12) are copied from the source array to the destination array. The 12 copied data elements of the source array start with index number 6, since SNX=6. The locations copied to in the destination array start with index number 8, since DNX=8. So %R0001, bit 6 through %R0002, bit 1 of the source array are read and then copied into %R0100, bit 8 through %R0101, bit 3 of the destination array.



Search Functions

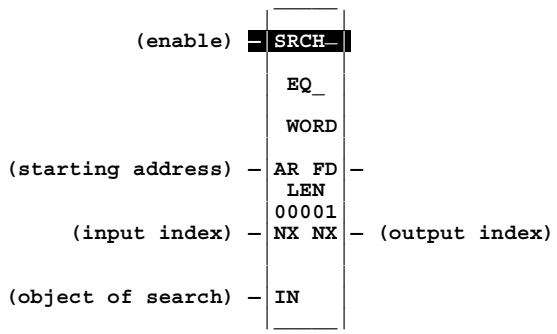
Use the appropriate Search function listed below to search for all array values for that particular operation.

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), its value is set to the default value of zero.



Parameters

Parameter	Description
enable	When the enable input is on, the operation is performed.
AR	AR contains the starting address of the array to be searched (the target array).
Input NX	Input NX contains an index number (in the target array) where the search is to begin.
IN	IN contains the object to be searched for.
Output NX	If the object of the search is found, its location in the array (its index number) will be written here.
FD	This output turns on to indicate that the searched for object has been found in the array.
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

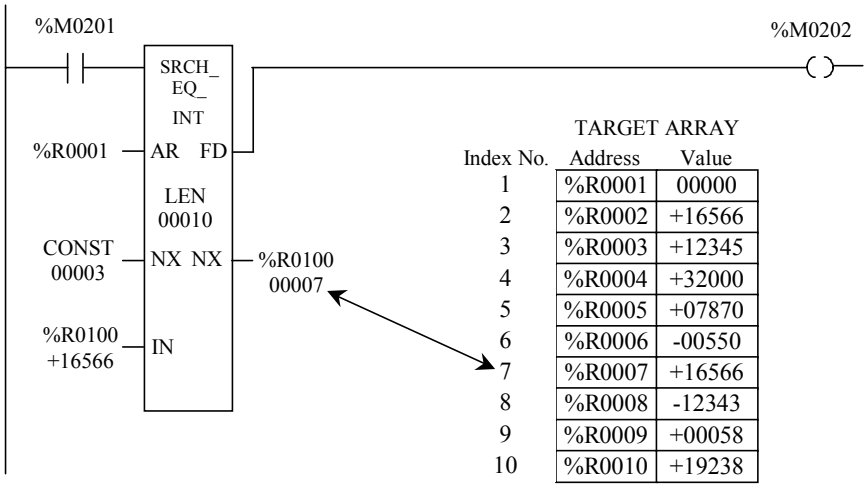
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
AR		o	o	o	o	Δ	o	•	•	•		
NX in		•	•	•	•		•	•	•	•	•	
IN		o	o	o	o	Δ	o	•	•	•	•	
NX out		•	•	•	•		•	•	•	•		
FD	•											•

• Valid reference or place where power may flow through the function.
o Valid reference for INT, BYTE, or WORD data only; not valid for DINT.
Δ Valid reference for BYTE or WORD data only; not valid for INT or DINT.

Example 1

The SRCH_EQ function (INT type) in this example searches the block of memory that starts at %R0001 (specified at AR) and continues through %R0010 (LEN=10). The value to be searched for, defined at IN, is +16566. Input NX, with a value of 3, indicates that the search is to begin at the fourth data element in the array since the NX value is incremented by 1 when the function executes.

When enable contact %M0201 is on, the SRCH_EQ function searches the specified array, starting at index number 4, for a value equal to the value at IN, +16566. It finds this value in %R0007, which has an index number of 7, so it writes the number 7 into the output NX at %R0100. It also turns on output FD, which indicates that it found the search object in the array. Note that although address %R0002 also contains the searched-for value of +16566, this data element was not included in the search because the input NX parameter value of 3 specified that the search start with the fourth data element, which is %R0004.



TARGET ARRAY

Index No.	Address	Value
1	%R0001	00000
2	%R0002	+16566
3	%R0003	+12345
4	%R0004	+32000
5	%R0005	+07870
6	%R0006	-00550
7	%R0007	+16566
8	%R0008	-12343
9	%R0009	+00058
10	%R0010	+19238

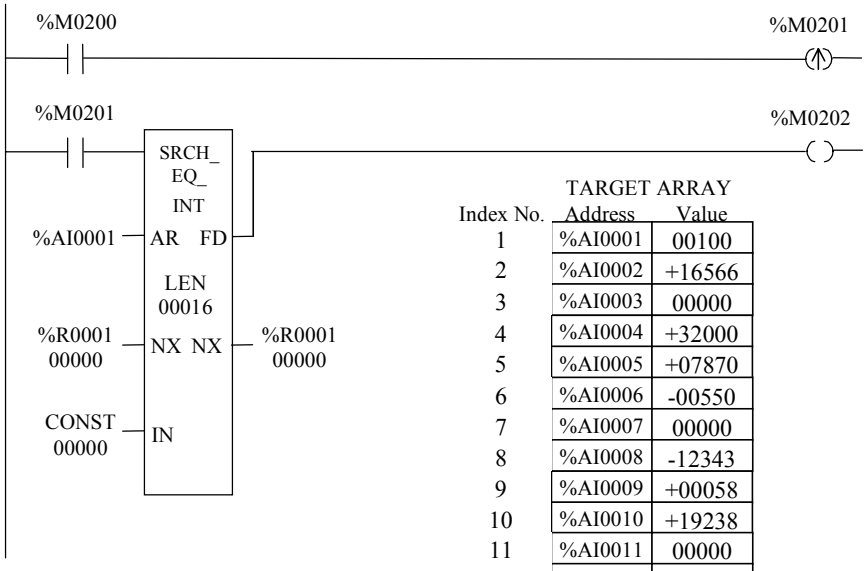
Example 2

The array in this example starts at %AI0001 (specified at AR) and continues through %AI0016 (LEN=16). The value to be searched for, defined at IN, is +16566. The input NX, with a starting value of 0, indicates that the search is to begin at the first data element in the array since the NX value increments by 1 when the function executes.

When %M0200 closes for the first time, the function executes its first search, starting with data element 1, for a value equal to the value at IN, 00000. It finds this value in %AI0003, which has an index number of 3, so it writes the number 3 into the output NX and input NX, which both have the reference address of %R0001. It also turns on output FD, which indicates that it found the search object in the array.

When %M0200 closes the second time, the input NX value, which is now set to 3, increments by 1, so the second search begins at the fourth array element, %AI0004. The target value of 00000 is now found in %AI0007, the seventh data element, so the number 7 is written to %R0001. Each succeeding search follows this pattern, until the fifth search, in which no target is found. Since no target is found, a 0 is written to %R0001, which will ensure that the search will start at the beginning of the array the next time the search is initiated.

Search No.	Search Starts at Data Element	Search Results (in %R0001)
1	1	3
2	4	7
3	8	11
4	12	15
5	16	0



TARGET ARRAY

Index No.	Address	Value
1	%AI0001	00100
2	%AI0002	+16566
3	%AI0003	00000
4	%AI0004	+32000
5	%AI0005	+07870
6	%AI0006	-00550
7	%AI0007	00000
8	%AI0008	-12343
9	%AI0009	+00058
10	%AI0010	+19238
11	%AI0011	00000
12	%AI0012	+16566
13	%AI0013	+12345
14	%AI0014	+32000
15	%AI0015	00000
16	%AI0016	-00550

Chapter 11

Conversion Functions

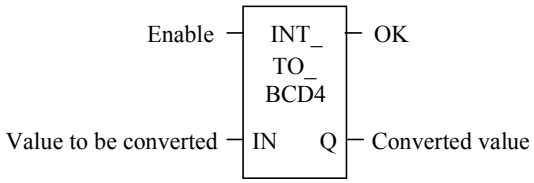
Use the conversion functions to convert a data item from one number type to another. Many programming instructions, such as math functions, must be used with data of one type. This section describes the following conversion functions:

Abbreviation	Function	Description	Page
BCD-4	Convert to BCD-4	Convert a signed integer to 4-digit BCD format.	11-2
INT	Convert to Signed Integer	Convert BCD-4 or REAL to signed integer.	11-3
DINT	Convert to Double Precision Signed Integer	Convert REAL to double precision signed integer format.	11-5
REAL	Convert to REAL	Convert INT, DINT, BCD-4, or WORD to REAL.	11-7
WORD	Convert to WORD	Convert REAL to WORD format.	11-9
TRUN	Truncate	Round the real number toward zero.	11-11

—>BCD-4 (INT)

The Convert to BCD-4 function is used to output the 4-digit BCD equivalent of signed integer data. The original data is not changed by this 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.



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	Output Q contains the BCD-4 form of the original value in IN.

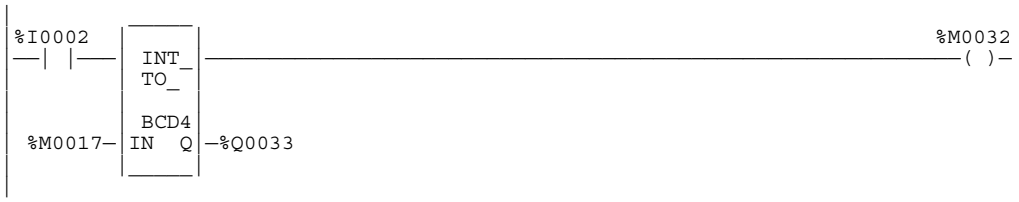
Valid Memory Types

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

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

Example

In the following example, when input %I0002 is set and no errors exist, the integer at input location %M0017 through %M0032 is converted to four BCD digits, and the result is stored in memory locations %Q0033 through %Q0048. Coil %M0032 turns on to verify successful conversion.



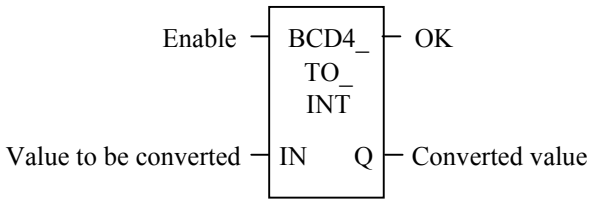
—>INT (BCD-4, REAL)

The Convert to Signed Integer function is used to output the integer equivalent of BCD-4 or REAL data. The original data is not changed by this function.

Note

The REAL data type is only available on 35x and 36x series CPUs, Release 9 or later, and on all releases of CPU352 and CPU37x.

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.



Parameters

Parameter	Description
enable	When the enable input is on, the conversion is performed.
IN	IN contains a reference for the BCD-4, REAL, or Constant value to be converted to integer.
ok	The ok output is energized whenever enable is energized, unless the data is out of range or NaN (Not a Number).
Q	Output Q contains the integer form of the original value in IN.

Valid Memory Types

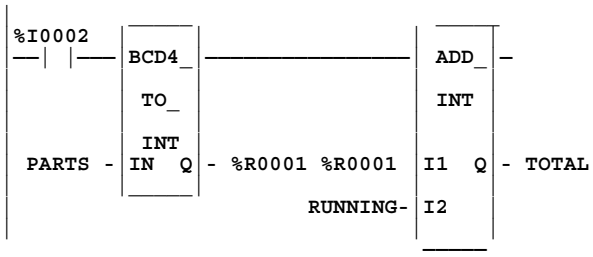
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•		•	•	•	•	•	
ok	•											•
Q		•	•	•	•		•	•	•	•		

Note: For REAL data, the only valid types are %R, %AI, and %AQ.

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

Example 1 – BCD4 to Integer

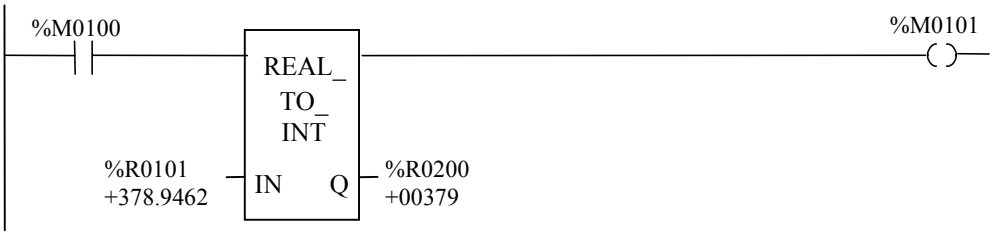
In the following example, whenever input %I0002 is set, the BCD-4 value in PARTS is converted to a signed integer and placed in %R0001. In the following ADD function, %R0001 is added to the signed integer value represented by the reference RUNNING. The sum is output by the ADD function to the reference TOTAL.



Example 2 – Real to Integer

This example shows conversion of a real number at %R0101 to an integer number at %R0200. When the enable input contact %M0100 is on, the conversion takes place. Note that during the conversion, the real number is rounded to the nearest integer. If the decimal portion of the real number is 0.5 or greater, the resulting integer is rounded up by a value of 1. If the decimal portion of the real number is less than 0.5, this decimal portion is discarded and the integer number is not rounded up. In the example below, real value 378.9462 is rounded up to integer value 379.

If rounding is not wanted, use the REAL_TRUN_INT function, which truncates the decimal portion of the real number, regardless of its value, during the conversion.



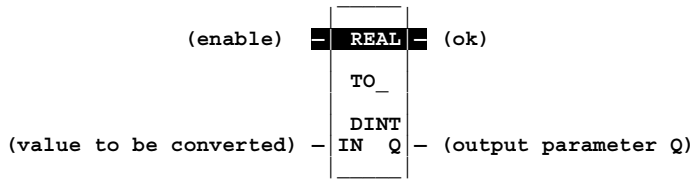
—>DINT (REAL)

The Convert to Double Precision Signed Integer function is used to output the double precision signed integer equivalent of real data. The original data is not changed by this function.

Note

The REAL data type is only available on 35x and 36x series CPUs, Release 9 or later, and on all releases of CPU352 and CPU37x.

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.



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.

Note

It is possible for a loss of precision to occur when converting from REAL to DINT since the REAL has 24 significant bits.

Valid Memory Types

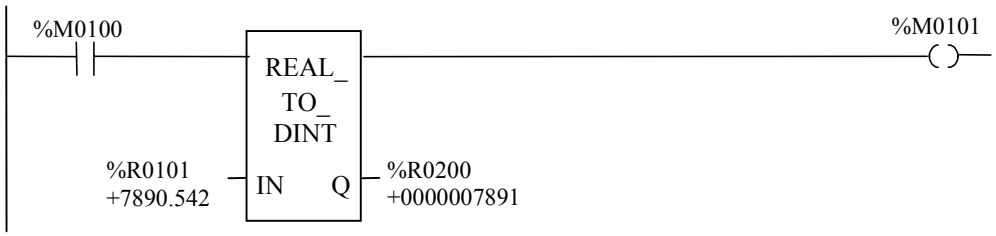
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		o	o	o	o		o	•	•	•	•	
ok	•											•
Q								•	•	•		

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

Example

In the following example, whenever enable input %M0100 is on, the real value at input location %R0101 is converted to a double precision signed integer, and the result is placed in location %R0200. Note that during the conversion, the real number is rounded to the nearest integer. If the decimal portion of the real number is 0.5 or greater, the resulting integer is rounded up by a value of 1. If the decimal portion of the real number is less than 0.5, this decimal portion is discarded and the integer number is not rounded up. In the example below, real value 7890.542 is rounded up to double integer value 7891.

If rounding is not wanted, use the REAL_TRUN_DINT function, which truncates the decimal portion of the real number, regardless of its value, during the conversion.



—>REAL (INT, DINT, BCD-4, WORD)

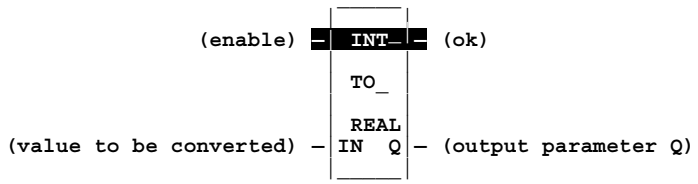
The Convert to Real function is used to output the real value of the input data. The original data is not changed by this function.

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 to REAL since the number of significant bits is reduced to 24.

Note

This function is only available on 35x and 36x series CPUs, Release 9 or later, and on all releases of CPU352 and CPU37x.



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.

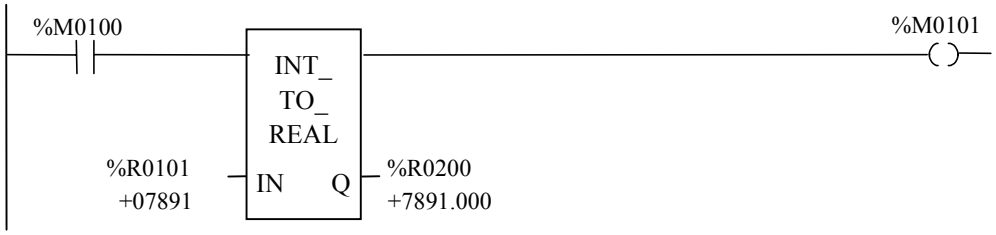
Valid Memory Types

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

• Valid reference or place where power may flow through the function.
o Not valid for DINT_TO_REAL.

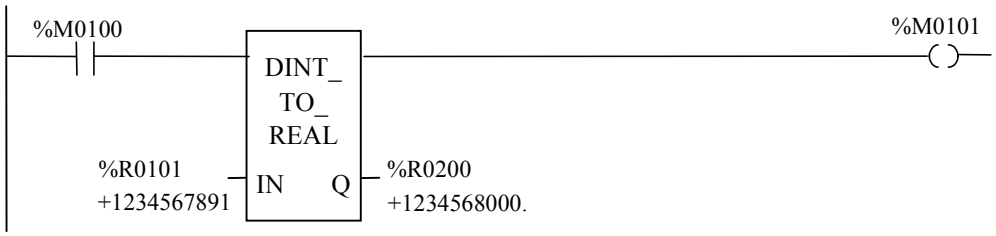
Example 1 - Integer to Real Conversion

In the following example, the integer value of input IN is +07891. The resulting value placed in %R0200 after the conversion to real format is +7891.000.



Example 2 – Double Integer to Real Conversion

In the following example, the double integer value of input IN is +1234567891. The resulting value placed in %R0200 after the conversion to real format is +1234568000. Note that a double integer number has 10 significant places, but a real number has only 7 significant places; therefore, an integer number is rounded to 7 significant places during the conversion to a real number. In the example shown, the four least significant digits, 7891, of the double integer number are rounded to 8000 in the four least significant digits of the real number.



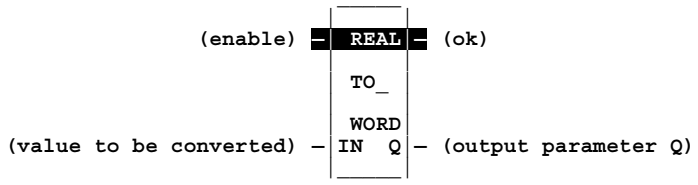
—>WORD (REAL)

The Convert to WORD function is used to output the WORD equivalent of real data. The original data is not changed by this function.

Note

This function is only available on the 35x, 36x, and 37x series CPUs.

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 FFFFh.



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 WORD.
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	%R	%AI	%AQ	const	none
enable	•											
IN								•	•	•	•	
ok	•											•
Q		•	•	•	•		•	•	•	•		

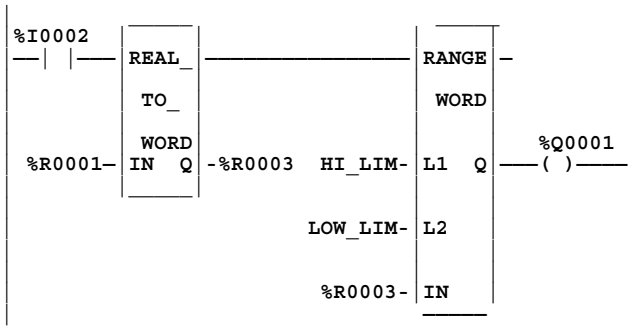
• Valid reference or place where power may flow through the function.

Example – Real to Word Conversion

In this example, since the RANGE function is not available as a REAL type, the real value in %R0001 is first converted to a word value (at %R0003), which is then used as the input to the following RANGE WORD function.

The table below shows the values at the various inputs and outputs for the following figure.

Item	Value or State
%R0001	15767.83
%R0003	3A89h (15,768 decimal)
HI LIM	4E20h (20,000 decimal)
LOW LIM	2710h (10,000 decimal)
Q1	ON



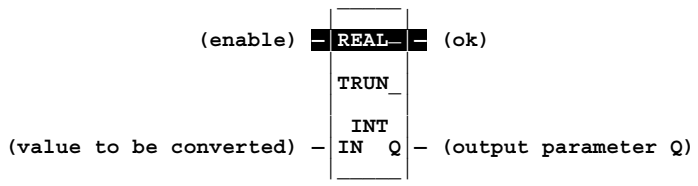
TRUN (INT, DINT)

The Truncate function is used to round a real number toward zero. During the conversion, all numbers to the right of the decimal place are discarded in the output number. The original number is not changed by this function.

Note

The 35x and 36x series CPUs (Release 9.00 or later and all releases of CPU352), and 37x are the only Series 90-30 CPUs with floating point capability; therefore, the TRUN function has no applicability for other 90-30 CPUs.

When the function receives power flow, it performs the conversion, making the result available via output Q. For CPU 352, 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). For all other 35x and 36x/37x series CPUs, the function does *not* pass power.



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.

Note

It is possible for a loss of precision to occur when converting from REAL to DINT since the REAL has 24 significant bits.

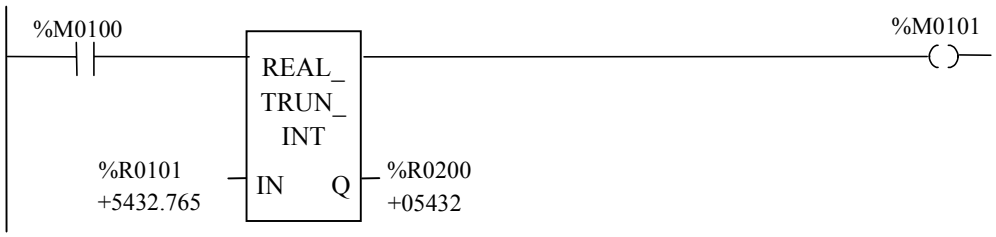
Valid Memory Types

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

• Valid reference or place where power may flow through the function.
o Valid for REAL_TRUN_INT only.

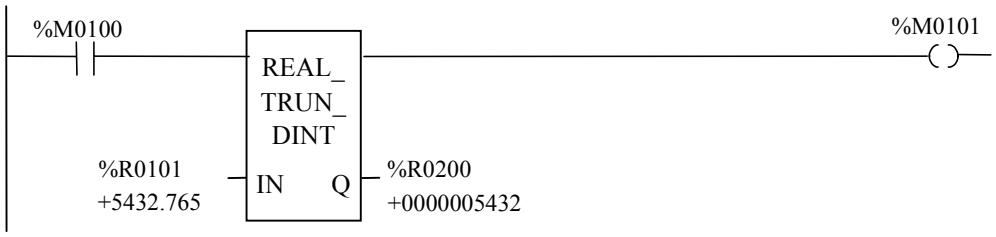
Example 1 – Truncate Real to Integer with Output Coil for CPU352

In the following example, the value at %R0101 is truncated (the decimal portion is discarded) and the resulting integer value of +05432 is placed into %R0200. . If a CPU352 were used, %M0101 would turn on, indicating a successful conversion. If any other 35x, 36x, or 37x CPU is used, no power flow is produced at the OK output, so no output coil would be programmed.



Example 2 – Truncate Real to Double Integer with Output Coil for CPU352

In the following example, the value at %R0101 is truncated (the decimal portion is discarded) and the resulting double integer value of +0000005432 is placed into %R0200. If a CPU352 were used, %M0101 would turn on, indicating a successful conversion. If any other 35x, 36x, or 37x CPU is used, no power flow is produced at the OK output, so no output coil would be programmed.



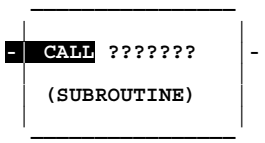
Chapter 12 Control Functions

This chapter describes the control functions, which can be used to limit program execution and alter 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 subroutine block.	12-2
DOIO	For one sweep, immediately services a specified range of inputs or outputs. (All inputs or outputs on a module are serviced if any reference locations on that module are included in the DO I/O function. Partial I/O module updates are not performed.) Optionally, a copy of the scanned I/O can be placed in internal memory, rather than the real input points.	12-3
SER	Sequential Event Recorder— collects a series of samples. A function control block contains user-supplied configuration of function block execution, sample configuration and operation parameters.	12-8
END	Provides a temporary end of logic. The program executes from the first rung to either the last rung or the END instruction, whichever is encountered first. This instruction is useful for debugging purposes, but it is not permitted in SFC programming (refer to the Note on page 12-8).	12-23
MCR and MCRN	Programs a Master Control Relay. An MCR causes all rungs between the MCR and its subsequent ENDMCR to be executed without power flow. Logicmaster 90-30/20/Micro software supports two forms of the MCR function, a nested form (MCRN) and a non-nested form (MCR).	12-24
ENDMCR and ENDMCRN	Indicates that the subsequent logic is to be executed with normal power flow. Logicmaster 90-30/20/Micro software supports two forms of the ENDMCR function, a nested form (ENDMCRN) and a non-nested form (ENDMCR).	12-30
JUMP and JUMPN	Causes program execution to jump to a specified location (indicated by a LABEL, see below) in the logic. Logicmaster 90-30/20/Micro software supports two forms of the JUMP function, a non-nested form (JUMP) and a nested form (JUMPN).	12-31
LABEL and LABELN	Specifies the target location of a JUMP instruction. Logicmaster 90-30/20/Micro software supports two forms of the LABEL function, a non-nested form (LABEL) and a nested form (LABELN).	12-33
COMMENT	Places a comment (rung explanation) in the program. After programming the instruction, the text can be typed in by "zooming" into the instruction (use F10 key to zoom).	12-34
SVCREQ	Requests a special PLC service. (See list of service requests on page 12-35.)	12-35
PID	Provides two PID (proportional/integral/derivative) closed-loop control algorithms: • Standard ISA PID algorithm (PIDISA). • Independent term algorithm (PIDIND).	12-70

CALL

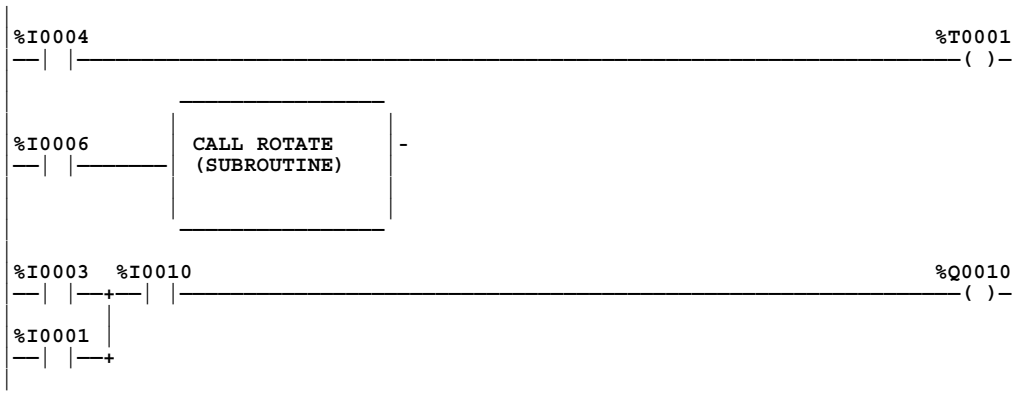
Use the CALL function to cause program execution to go to a specified subroutine block.



When the CALL function receives power flow, it causes the scan to go immediately to the designated subroutine block and execute it. After the subroutine block execution is complete, control returns to the rung in the logic immediately following the CALL instruction.

Example

In the following example, the CALL instruction is programmed to call the subroutine named ROTATE when contact %I0006 is on. (Note that before you can enter a subroutine name in a CALL instruction, the subroutine name must already exist in the Block Declarations table.) By positioning the cursor within the CALL instruction, you can press **F10** to zoom into the subroutine to view the subroutine logic. Once a subroutine is called, program execution will branch to the subroutine, which will execute to completion, then pass program execution over to the rung following the calling rung. In the example below, the subroutine is called from the second rung, so when the subroutine finishes executing, the program scan will resume with the third rung.



Note

Micro PLCs do not accommodate subroutines; therefore, the CALL function is inappropriate for use with a Micro PLC.

DOIO

The DO I/O (DOIO) function is used to update specified inputs or outputs for one scan while the program is running. The DOIO function can also be used to update selected I/O during the program in addition to the normal I/O scan. Under normal circumstances, the input tables are updated during the input scan portion of the PLC sweep and will not be updated again until the next sweep. The output tables are updated during the logic solution portion of the PLC sweep, but the output modules are not updated until the logic solution portion is finished. With the DO I/O function, updates of the input tables and output modules can be forced during the logic solution portion of the scan. This capability allows you to read input changes and write to outputs more quickly than is possible with the normal PLC scan. Refer to Chapter 2 for more information about the PLC sweep.

If input references are specified, the function allows the most recent values of inputs to be obtained (written to the input tables) for program logic. If output references are specified, DO I/O updates output modules based on the most current values stored in I/O memory. I/O is serviced in increments of entire I/O modules; the PLC adjusts the references, if necessary, while the function executes.

Use with Input Modules

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 applicable input table is 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 applicable input table is updated.

Use with Output Modules

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 modules. Program execution then returns to the next function following the DO I/O.

Use with Option Modules

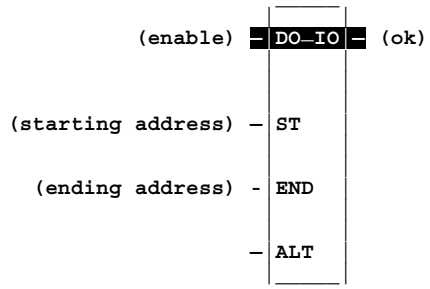
If the range of references includes an option module (HSC, APM, etc.), then all of the input data (%I and %AI) or all of the output data (%Q and %AQ) for that module will be scanned. The ALT parameter is ignored while scanning option modules. Also, if it is desired to use the DOIO with an Enhanced GCM module (IC693CMM302), the requirement in the following note must be met.

Note

The DOIO function ***can only*** be used with an Enhanced GCM module (IC693CMM302) in systems with Release 9.0 and later CPUs.

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” fault.



Parameters

Parameter	Description
enable	When the enable input is on, a limited input or output scan is performed.
ST	ST is the starting address of a group of input or output points or words to be serviced.
END	END is the ending address of the group 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. For Model 331 and later CPUs, the ALT parameter can have an effect on speed of DOIO function block execution (see Note below and the section on the enhanced DO I/O function for 331 and later CPUs later in this chapter). **If the ALT function is not used, this input should be left blank; if a constant value of 0 is programmed for ALT, the CPU may experience Watchdog Timeout Errors.**
ok	The ok output is energized when the input or output scan completes normally.

Note

An Enhanced DOIO function is available for Model 331 and later CPUs. In the Enhanced DOIO, the ALT parameter can be used to enter the slot number of a single discrete input or output module in the main rack. This Enhanced DOIO function will execute in 80 microseconds instead of the 236 microseconds required when the DOIO is programmed without the ALT parameter. No error checking is performed to prevent overlapping reference addresses or module type mismatches. See the “Enhanced DO I/O Function” section later in this chapter for details.

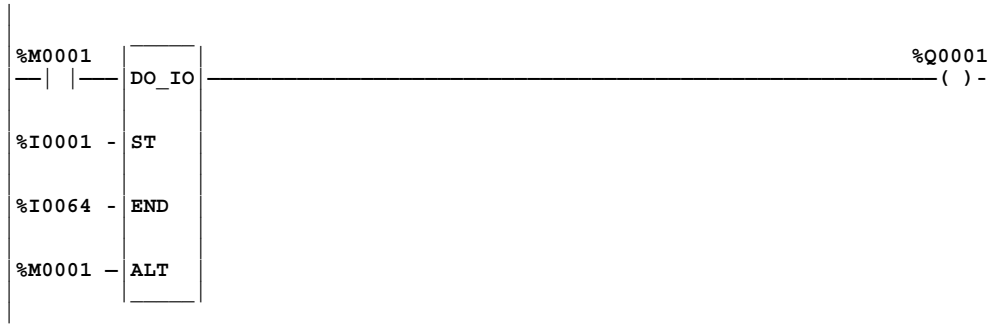
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
ST		•	•						•	•		
END		•	•						•	•		
ALT		•	•	•	•		•	•	•	•		•
ok	•											•

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

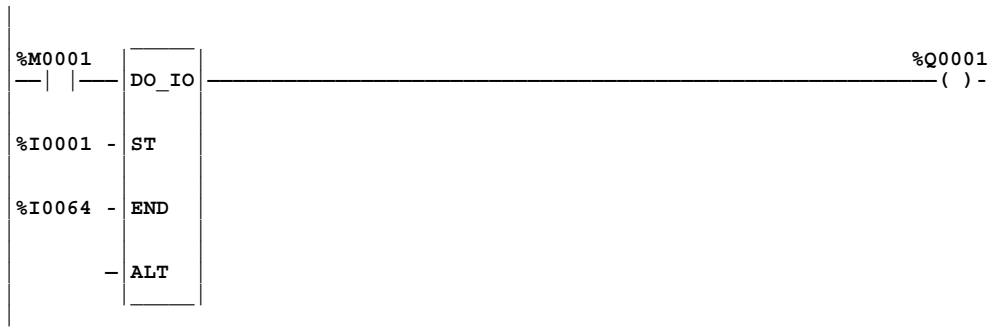
Input Example 1

In the following example, when enabling input %M0001 turns ON, references %I0001 (specified at ST) through %I0064 (specified at END) are scanned and %Q0001 is turned on. A copy of the scanned inputs is placed in internal memory from reference %M0001 (specified at ALT) through %M0064. Because an alternate location was specified at ALT, the %I input table is not updated by the DO_IO. This form of the function can be used to compare the current values of input points with their previous values (i.e. their values at the beginning of the logic solution scan).



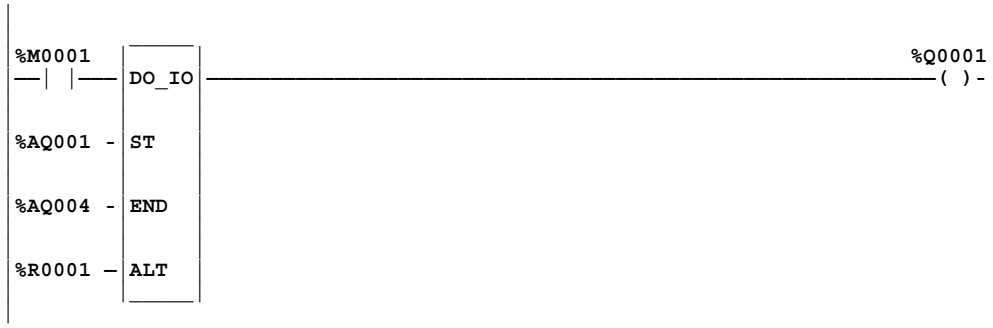
Input Example 2

In the following example, when enabling input %M0001 is ON, references %I0001 (specified at ST) through %I0064 (specified at END) are scanned and %Q0001 is turned on. Since no alternate memory location is specified at ALT, the scanned input values are used by the DO_IO to update the input table from reference %I0001 to %I0064. This form of the function allows input points to be scanned and updated one or more times during the logic solution portion of the CPU sweep. Note that when the ALT input is not used, it should be left blank as shown. Do not place a zero on the ALT input because that will cause Watchdog Timer faults.



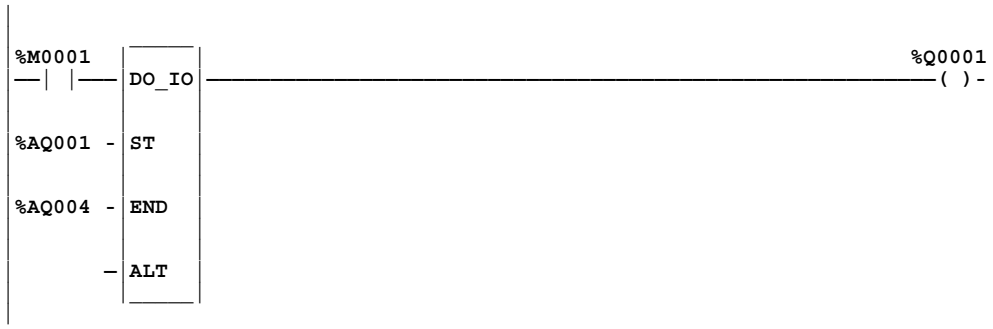
Output Example 1

In the following example, when enabling input %M0001 is ON, the values of analog output channels %AQ001 (specified at ST) through %AQ004 (specified at END) are written to references %R0001 (specified at ALT) through %R0004 respectively, and %Q0001 is turned on. Because the %R0001 alternate location was specified at ALT, the values at %AQ001 through %AQ004 are not written to the analog output modules by the DO_IO.



Output Example 2

In the following example, when the enabling input %M0001 is ON, the values at references %AQ001 through %AQ004 are written to analog output channels %AQ001 through %AQ004 on the applicable analog output modules, and %Q0001 is turned on. The DO_IO updates the analog output modules because no alternate memory location is specified at ALT. Note that when the ALT input is not used, it should be left blank as shown. Do not place a zero on the ALT input because that will cause Watchdog Timer faults.



Enhanced DO I/O Function for 331 and Later CPUs

Caution

Programs containing an Enhanced DO I/O should not be loaded by a version of Logicmaster 90-30/20 software earlier than 4.01.

An enhanced version of the DO I/O (DOIO) function is available for Release 4.20 or later, of Models 331 and later CPUs. This enhanced version of the DOIO function can only be used on a single discrete input or discrete output 8-point, 16-point, or 32-point module.

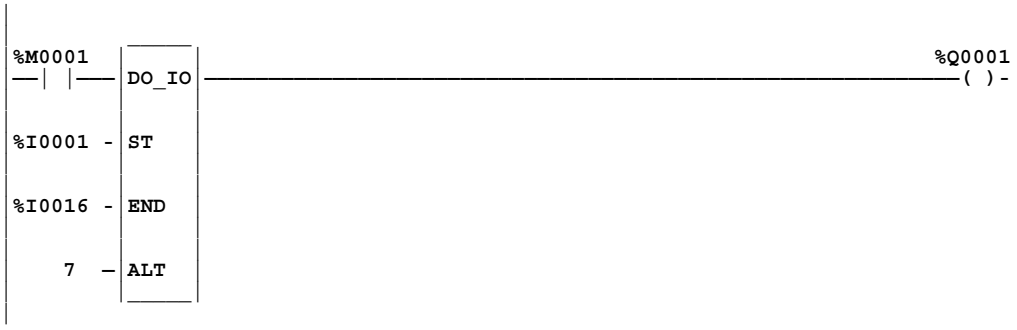
The ALT parameter identifies the slot in the main rack of the target module. For example, a constant value of 2 at ALT indicates that the module in slot 2 is targeted. The ST and END parameters set the range of memory to be acted upon.

Note

The only checking associated with the enhanced DOIO function block is a basic check of the target module's condition.

The enhanced DOIO function only applies to modules located in a modular CPU rack. Therefore, the ALT parameter must be between 2 and 5 for a 5-slot rack or 2 and 10 for a 10-slot rack.

The start (ST) and end (END) references must be either %I or %Q. These references specify the first and last reference the module is configured for. For example, if a 16-point input module is configured at %I0001 through %I0016 in slot 7 of a 10-slot main (CPU) rack, the ST parameter must be %I0001, the END parameter must be %I0016, and the ALT parameter must be 10, as shown below:



The following table compares the execution times of a normal DOIO function block for an 8-point, 16-point, or 32-point discrete input/output module with those of an enhanced DOIO function block.

Module	Normal DOIO Execution Time	Enhanced DOIO Execution Time
8-Pt Discrete Input Module 8-Pt Discrete Output Module	224 microseconds 208 microseconds	67 microseconds 48 microseconds
16-Pt Discrete Input Module 16-Pt Discrete Output Module	224 microseconds 211 microseconds	68 microseconds 47 microseconds
32-Pt Discrete Input Module 32-Pt Discrete Output Module	247 microseconds 226 microseconds	91 microseconds 50 microseconds

SER (Sequential Event Recorder)

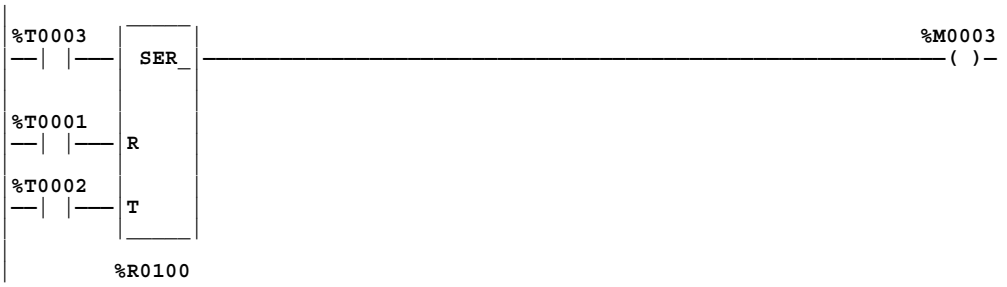
Requires CPUs 35x or 36x with Firmware 9.00 or later, or CPU37x

- The SER (Sequential Event Recorder) function block collects a series of discrete samples (it only works with discrete data). An SER function block collects up to 32 contiguous or non-contiguous bits per sample when the Enable input receives power flow.
- Each SER can capture up to 1024 samples, with up to 32 bits per sample.
- If the SER function block is embedded in a periodic subroutine, sampling rate is based on the periodic subroutine execution rate.
- Only the trigger sample is time stamped. The trigger sample can be time-stamped in BCD (maximum resolution is 1second) or POSIX format (maximum resolution is 10ms). The time stamp is only placed once at the trigger point. The SER does not support more than one time stamp per recording.
- The SER can be configured for pre-, mid-, or post-trigger modes. (See page 12-14.)
- SER operation is configured by a function control block that you can create using a series of Block Move (BLKMV) commands. (See page 12-10.)
- An input module may be optionally specified that will be scanned each time the SER executes. This helps ensure that the data captured from the specified module is as up-to-date as possible.

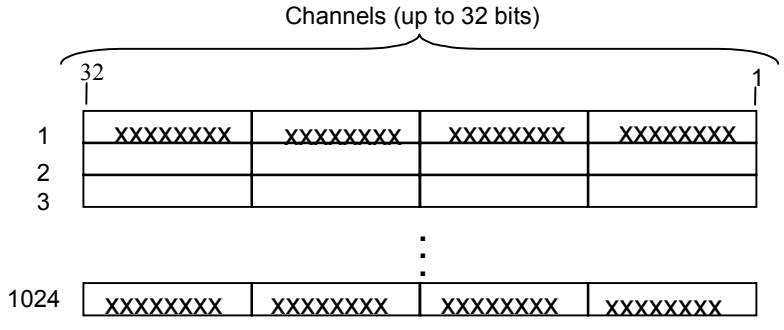
Note

PLC-to-PLC synchronization is not supported.

The SER function block has one output and three inputs: enable, reset (R), and trigger (T).



As shown below, 8, 16, 24, or 32 channels may be configured, with each channel representing a discrete point. Also, up to 1024 samples may be specified.



Parameters

Parameter	Description
enable	Whenever the function is enabled and the reset input is off, the SER function block collects one sample from all configured channels.
R	When the reset input receives power flow, the SER function is reset regardless of the state of the enable input. Sample Buffer, Trigger Sample Offset, Trigger Time, and Current Sample Offset are all cleared to zero. The function block remains in the reset state until power flow is removed from the reset input. The OK output is turned off while in the reset state. When the power flow is removed from the reset input, sampling resumes.
T	If the Trigger Input mode is selected and the function block is enabled, when the trigger input goes on, the SER to transition to the triggered state. The Trigger Time, Trigger Sample Offset, and a data sample are recorded. The trigger sample will be recorded regardless of the number of samples taken. Once triggered, the event recorder continues sampling until the Number of Samples After Trigger is satisfied, at which time it stops collecting samples until power flow is seen on the reset input. If Trigger Mode is set to Full Buffer, the trigger signal is ignored. For information on configuring Trigger Mode, see “Function Control Block” on page 12-10.
Starting Reference	The 78-word function control block array begins at this reference. The function control block defines function block execution, sample configuration, and operation parameters. For details, see “Function Control Bloc” on page 12-10
ok	The ok output is energized when the trigger conditions are satisfied (specified by the Trigger Mode parameter), and all sampling is complete. The output continues to receive power flow regardless of the state of the enable input until the reset receives power flow.

Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
Control Block								•				
R	•											
T	•											
ok	•											•

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

Function Control Block

The function control block is a 78-word array that defines information about the data capture and trigger mechanism for the SER function. In a particular program, only one Sequential Event Recorder function block can be associated with each function command block and data block.

Perform the following steps to configure parameters for the SER function block:

1. Set up the stored values for the array as defined in the table below. You can use block moves to initialize the registers, or initialize the data in the register table and store the table before activating the SER function.
2. Add the SER function block to your ladder logic.

Note

If you require x channels where x is not equal to 8, 16, 24, but is less than 32, you must select a number of channels which is greater than x and a multiple of 8, and fill in a null channel description for the unused channels. A null channel description has a segment selector of 0xFFh, a length parameter equal to the number of unused channels, and a 0 offset.

Word	Parameter	Description
0 (starting reference)	Status	Read only variable that indicates the current state of the SER function block. Additional information is provided in Status Extra Data, (Word 1). **Note:** If an error is detected in the Control Block, the status will be set to 6, the OK output will be cleared, and no action will occur. Settings for Status include: 0 = Reset 1 = Inactive 2 = Active 3 = Triggered 4 = Complete 5 = Overrun Error 6 = Parameter error
1	Status Extra Data	A read-only variable that provides additional state information about the SER function. See “Status Extra Data States” on page 12-12 for settings for this parameter.
2	Trigger Mode	Defines conditions for the SER function block to transition to the triggered state. Valid settings are: 0 = Trigger Input mode 1 = Full Buffer mode In Trigger Input mode, if the function block is enabled, a time stamp is generated when the Trigger signal is activated. Sampling continues until the Number of Samples After Trigger value has been satisfied. When this occurs, the OK output is activated. In Full Buffer mode, the Trigger signal is ignored. When the function block is enabled, sampling continues until the sample buffer is filled. When this happens the OK output is activated. The Number of Samples parameter sets buffer size.
3	Trigger Time Format	Determines how the Trigger Time will be displayed. For BCD display, set this parameter to 0. For POSIX display, set this parameter to 1. (For details, see page 12-17.)
4—7	Reserved	Words 4 through 7 are reserved and should be set to zero.

Word	Parameter	Description
8	Number of Channels (bits per sample)	Specifies the number of bits of data that will be sampled and copied to the sample buffer for each execution of the function block. Valid choices are 8, 16, 24, or 32 bits. Any unused channels must be configured with a null channel description. (See Words 14—77.) For example, if 19 bits are needed, you must configure 24 and specify that the last five are null.
9	Number of Samples	Specifies the sample buffer size. Valid choices are 1 to 1024 samples. (Actual buffer size in bits is Number of Samples times Number of Channels.)
10	Number of Samples After Trigger	Specifies the number of samples that are collected after the trigger condition becomes true. This parameter can be set to a value between 0 and the Number of Samples. This parameter is valid only when the Trigger Mode is set to zero (Trigger Input).
11	Input Module Slot	Specifies the location in the main rack (Rack 0) of an input module that will be scanned each time the SER executes. If the value is 0, no module is scanned. When an input module is scanned, its values are stored locally and the values of the reference addresses configured for the module are not affected. To store values from the scanned input module into the data block sample buffer, a channel description must be provided. If the module is not present or is faulted at the time of the scan, the data returned will be zero. A fault will not be logged in the fault table if this occurs; fault indication will be left to the IO scanner.
12	Data Block Segment Selector (Memory Type)	Specifies the data type allocated for the Data Block. For example, if you wanted use the %R memory type, you would enter 08 for this parameter. Valid settings for this parameter include: %R (08h), %AI (0Ah), %AQ (0Ch). For details on the data block, see page 12-13.
13	Data Block Offset	Specifies the starting reference for the Data Block. This parameter is zero based. For example, if you wanted to begin at %R0100, you would enter 99 for this parameter. Be sure to allow enough memory for the entire data block.
14—77	Channel Descriptions	Specifies the reference location (Segment Selector, Length and Offset) associated with a particular channel. There can be from 1 to 32 channel descriptions, depending upon the number of channels being sampled and data length. Data is returned in the order defined in this section.
	Channel Segment Selector/Length	Entered as a hexadecimal value, this word defines the segment selector and data length (in bits). MSB = Segment Selector. LSB = Data Length. The data length is useful for samples that are contiguous. The Segment Selector can be set to any discrete data type: %I (46h), %Q (48h), %M (4Ch), %T (4Ah), %G (56h), %S (54h), %SA (4Eh), %SB (50h), %SC (52h), Null Selector (FFh), and Input Module Selector (00h).
		The length parameter can range from 1—32, but the sum of all of the lengths must not be greater than the Number of Channels parameter. A length greater than 1 allows multiple contiguous channels to be configured with a single channel description.
	Channel Offset	Entered as a hexadecimal value, this word defines the BIT offset for the data type or input module specified in the Segment Selector. This offset is zero-based. The range for this parameter varies, depending on the Segment Selector (data type and length). The offset indicates the location within the data table or input module at which to sample.

Status Extra Data States

The Status Extra Data (Word 1 in the function control block) provides additional state information for the SER function.

Value	State	Description
0	Reset State	The Reset input is receiving power flow. Sample Buffer, Trigger Sample Offset, Trigger Time, and Current Sample Offset are all cleared to zero. The output is held to no power flow. Transition to the ***Inactive State*** occurs when the reset power flow is removed. Status Extra Data has no significance and will be cleared to zero.
1	Inactive	State between the Reset State and the Active State. No actions are performed in this state. The SER output is held to no power flow. Transition to the Active State occurs when the function block receives enable power flow.
2	Active	The Enable input has received power flow, but the function block is not reset, in error, or triggered. One sample is recorded for each execution when the function block is enabled. The output is held to no power flow. The Trigger condition (specified by the Trigger Mode parameter) is monitored and will cause transition to the Triggered State if conditions are true. If more than the "Number of Samples" have been taken, Status Extra Data will be set to 0x01, otherwise it will be 0x00.
3	Triggered	State if the trigger condition defined by Trigger Mode is true. Additional Samples are taken depending upon the trigger mode and parameter settings. The output is held to no power flow. Transition to the Complete state will occur when all sampling is complete. If more than the "Number of Samples" have been taken, Status Extra Data will be set to 0x01, otherwise it will be 0x00.
4	Complete	All sampling is complete. The output receives power flow. Only transition to the Reset State is allowed. If more than the "Number of Samples" have been taken then Status Extra Data will be set to 0x01, otherwise it will be 0x00.
5	Overrun Error	The Control/Data Block has exceeded the end of its memory type. The output is held to no power flow. Only transition to the Reset State is allowed. Status Extra Data has no significance and will be cleared to zero.
6	Parameter Error	There is an error in the function control block or other operation parameters. The output is held to no power flow. Only transition to the Reset State is allowed. The Status Extra Data word contains the offset into the control block at which the parameter error occurred.
7	Status Error	The Status Parameter is invalid. The output is held to no power flow. Only transition to the Reset State is allowed. The invalid status value will be stored in the Status Extra Data location in the Control Block.

SER Data Block Format

The SER Data Block contains the sample buffer, sample offsets, and trigger information. This information is supplied by the CPU and you should only read from this data area. It is your responsibility to allocate enough register space for the Data Block. The block format is as follows:

Word*	Parameter Description
0	Current sample offset number. References the location where the most recent sample was placed. The parameter is zero-based. Valid ranges are –1 to 1023. Register Location of Sample = (Num Bytes per Sample) * (Offset Parameter)/2 + (Sample Buffer Starting Register). **Note:** This value is not valid until a sample is taken. This value is set to –1 when the SER function is reset through the Reset input.
1	Trigger sample offset number. References the storage location of the sample obtained when the trigger condition transitioned to the True state. The parameter is zero-based. Valid ranges are 0 to 1023. Register Location of Sample = (Num Bytes per Sample) * (Offset Parameter)/2 + (Sample Buffer Starting Register). **Note:** This value is not valid until the trigger condition is met. This value is set to 0 when the SER function is reset (through the reset input).
2 through 5	Trigger Time: Indicates the time, according to the Time of Day clock within the PLC, that the trigger condition transitioned to the true state within the function block. The time value can be displayed in BCD format (default) or POSIX format. The format is determined by the *Trigger Time Format* parameter in the Control Block. This value is initialized to zero upon activation of the reset input.
6 to end of sample buffer.	Sample Buffer. The area of memory that holds the data samples. This area is set to zero when the reset parameter is energized. The sample buffer size varies, depending on the number of channels and sample size. The sample buffer is a circular buffer – when the last location is written, the next sample will overwrite the sample in the first register. End of sample buffer = 5 + ({[(# of samples to be taken) * (# of channels to be sampled / 8)] +1} / 2

*Offset from starting reference defined by Data Block Segment Selector (Word 12) and Data Block Offset (Word 13) in Function Control Block.

SER Operation

If the SER is enabled when scanned, it reads the configured sample points and puts them in a circular list. After the configured number of samples is taken, the output is turned on. The transition of the output can be used to record the time that the last sample is taken or to initiate additional sampling. (See "Sampling Modes.")

The SER function block must be reset (enable the Reset input power flow) before sampling is started. Resetting initializes the data block area. If the function block status is not reset, it will execute with the current values in the data block, causing the current sample offset to be incorrect and invalid data in the data block.

The Control Block of the SER function block is scanned every time the function block is executed in the Reset, Active, or Triggered State. If you change a configuration parameter in the Control Block during program execution, the change takes effect the next time the SER function block associated with that Control Block is scanned. If an error is encountered, operation stops and the

function block goes to the appropriate error state. You must correct the error and then reset the function block (enable the Reset input power flow) to begin sampling again.

If you select an input module to be scanned the PLC will *not* verify that the module is a Discrete Input Module, or that Channel Descriptions associated with the module have valid lengths and offsets based upon the module size. You must correctly set up the sampling of an Input Module. Although multiple channel descriptions can target an input module, the module is still only scanned once per function block execution.

The SER function block can be placed in the normal user logic program or within a periodic subroutine. If placed in the user logic program, the resolution of the interval between scans is the resolution of the scan time, which can vary depending on the number and types of functions active on any particular scan. If placed in an interrupt subroutine, the interval can be set to as little as 1ms, and the resolution will be highly repeatable at 1ms with little jitter.

Execution time of one function block with a 1ms periodic subroutine can consume up to 50% of the CPU's resources. You should not plan on execution of more than two SER functions within a 1ms periodic subroutine.

Sampling Modes

The SER sampling mode is determined by the Trigger Mode (Word 2 in the Function Control Block) and Number of Samples After Trigger (Word 10) parameters. You will need to interpret the contents of the sample buffer based on how you configured these parameters.

The following table summarizes how the sampling modes are determined.

Mode	Word 2	Word 10
Pre-Trigger	0	0
Mid-Trigger	0	From 1 to (Number of Samples – 1)
Post-Trigger	0	equal to Number of Samples (specified in Word 9)
Full Buffer	1	Word 10 and trigger input signal are ignored

Trigger-Controlled Sampling

In order to configure pre-, mid-, and post-trigger sampling modes, Trigger Mode (Word 2 = 0) must be selected. The sampling mode is controlled by the Number of Samples After Trigger value (Word 10). In all cases, sampling starts when the Enable signal goes high. When the Trigger signal goes high, sampling continues until the number specified in the Number of Samples After Trigger parameter is collected. The SER's OK Output signal goes high when sampling is completed.

If more than the configured Number of Samples (Word 9) is collected before the Number of Samples After Trigger condition is satisfied, the buffer "wraps around," meaning that the SER returns to the beginning of the buffer and overwrites the initial samples.

When the trigger first transitions from off to on, the trigger time is placed in a configured location.

Pre-Trigger

Collects samples continuously until trigger is detected.

To configure this mode, set Word 10 to a value of 0, so that when the trigger signal is activated, sampling stops and a time stamp is generated. (All samples are collected before the trigger.)

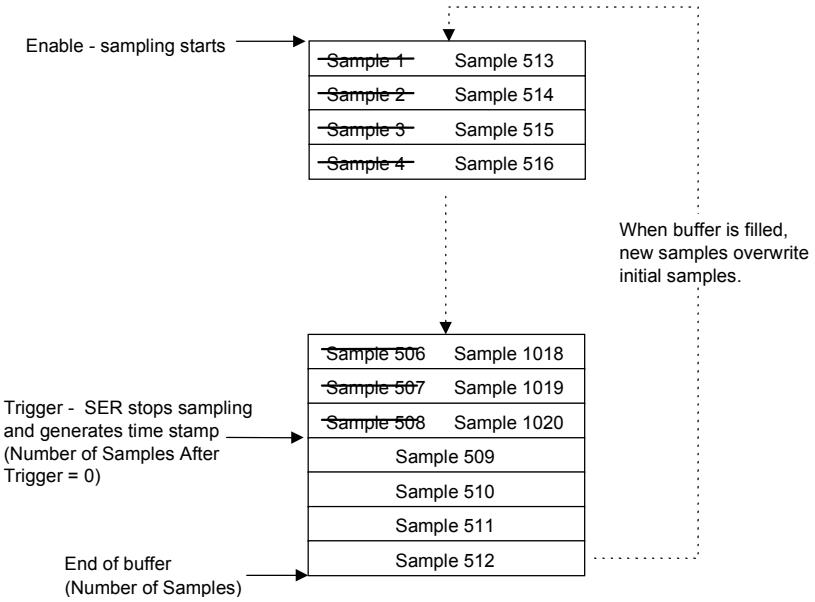


Figure 12-1. Example of Pre-Trigger SER Sampling (for 512 Samples)

Mid-Trigger

Collects samples continuously until Number of Samples After Trigger has been collected.

To configure this mode, set Word 10 to a value between 1 and the (Number of Samples – 1). When the trigger signal is activated, sampling continues until the configured number has been collected. In the following example, Number of Samples After Trigger is 12. When sampling is complete, the buffer will contain 500 pre-trigger samples and 12 post-trigger samples.

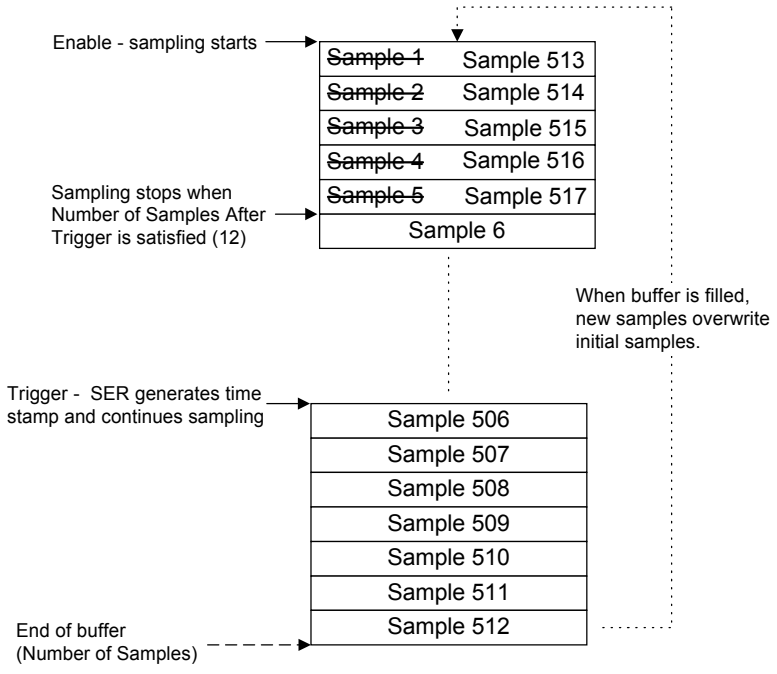


Figure 12-2. Example of Mid-Trigger SER Sampling (for 512 Samples)

Post-Trigger

Collects sample continuously until Number of Samples is reached.

To configure this mode, set Word 10 to a value equal to the Number of Samples (Word 9). When the trigger signal is activated, sampling continues until the configured number has been collected. (Note: all samples are collected after the trigger.)

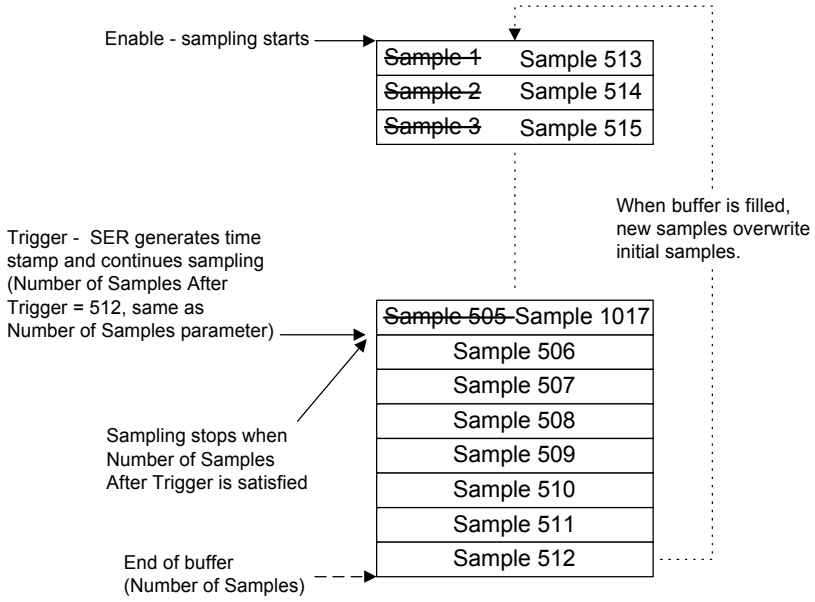


Figure 12-3. Post-Trigger SER Sampling (for 512 samples)

Full Buffer (Trigger Does Not Control Sampling)

If the Trigger Mode is set to 1, the Number of Samples After Trigger parameter (Word 10) is ignored and the Trigger input signal has no effect on function block operation. When the function block is enabled, sampling continues until the Number of Samples (Word 9) is collected, filling the sample buffer. When the buffer is full, sampling stops, a Trigger time stamp is generated, and the function block OK output goes high.

SER Function Block Trigger Timestamp Formats

BCD Format		
Data Block Word No.	**Contents (High Byte/Low Byte)**	**Suggested Viewing Format**
Word 2	Month/Year	Hex. (MMYY)
Word 3	Hours/Day of Month	Hex. (HHDD)
Word 4	Seconds/Minutes	Hex. (SSMM)
Word 5	Not Used	All zeros

POSIX Format		
Data Block Word No.	**Contents**	**Suggested Viewing format**
Words 2 and 3	Number of Seconds Since January 1, 1970	Dint
Words 4 and 5	Number of Nano-Seconds into next Second	Dint

Example

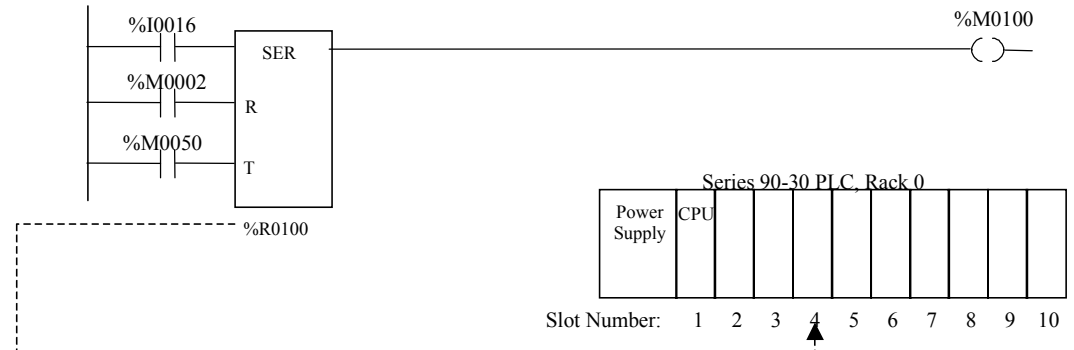
The next two tables show how the trigger time of November 3, 1998 at 8:34:05.010 a.m. would appear in BCD and in POSIX formats in a data block that starts at %R0201 (Word 0).

November 3, 1998 at 8:34:05.010 a. m. in BCD Format		
Register	**Parameter**	**Value (hex)**
%R0203	Month/Year	1198
%R0204	Hours/Day of Month	0803
%R0205	Seconds/Minutes	0534
%R0206	Unused	0000

November 3, 1998 at 8:34:05.010 a. m. in POSIX Format			
Register	**Parameter**	**Value (decimal)**	**Value (hex)**
%R0203/R0204	Seconds	910,082,045	363EBFFD
%R0205/R0206	Nano-seconds	010,000,000	00989680

SER Example

The following shows the interrelationships, of the ladder logic instruction, the control block in PLC memory, and the affected Input module in the PLC. The control block has been set up as described in Table 12-1.



Control Block

Word #	Address	Value	Description
0	%R0100	0	Status word
1	%R0101	0	Status extra data
2	%R0102	0	Trigger mode (0 = Trigger input)
3	%R0103	0	Trigger time format (0 = BCD)
4	%R0104	0	Reserved
5	%R0105	0	Reserved
6	%R0106	0	Reserved
7	%R0107	0	Reserved
8	%R0108	24	Bits per channel
9	%R0109	512	Sample buffer size
10	%R0110	12	Number of samples after trigger
11	%R0111	4	Specifies the input module in slot 4
12	%R0112	8	Data block memory type (%R)
13	%R0113	200	Data block memory offset (201)
14 ↓	%R0104 ↓		Channel Descriptions

%R0201

Data Block

Word #	Address	Value	Description
0	%R0201	0	Current sample offset number
1	%R0202	0	Trigger sample offset number
2	%R0203	0	Month and year (MMYY)
3	%R0204	0	Hour and day (HHDD)
4	%R0205	0	Seconds and minutes (SSMM)
5	%R0206	0	Not used
6 ↓	%R0207 ↓		Sample Buffer (size specified by Control Block Word 9)

Function Control Block Example

In this example, a 16-point discrete input module in rack 0, slot 4, has been specified (in Word 11) as the target to sample. It has been executing long enough that 572 samples (512 + 60) have been taken. The Enable input is receiving power flow, but the Reset and Trigger inputs are not.

Table 12-1. Function Control Block for SER Example

Word	Register	Parameter	Value (dec)	Value (hex)	Description
0	%R0100	Status	2	0002	Function block is in the Active state. This means the function block is executing normally, and taking a sample each time the function block is encountered in program logic.
1	%R0101	Status Extra Data	1	0001	The extra status data indicates that more that 512 samples have been taken and thus the sample buffer has already wrapped at least once.
2	%R0102	Trigger mode	0	0000	The event recorder is configured to trigger based on the Trigger input.
3	%R0103	Trigger Time Format	0	0000	0=BCD
4	%R0104	Reserved	0	0000	The Reserved parameters are always set to 0.
5	%R0105	Reserved	0	0000	
6	%R0106	Reserved	0	0000	
7	%R0107	Reserved	0	0000	
8	%R0108	# of channels	24	0018	Each sample consists of 24 bits (3 bytes) of data.
9	%R0109	# of samples to be taken	512	0200	Sample buffer size is 512 samples. Note that the sample buffer equals 512 x (24/8) = 1536 bytes or 768 words. (Each sample is 3 bytes long as specified in Word 8 above.)
10	%R0110	# of samples after trigger	12	000C	The number of samples to be collected after the trigger occurs is 12.
11	%R0111	Input module slot	4	0004	The input module in rack 0, slot 4 will be scanned when the SER executes so that its current values are available for sampling by the SER.
12	%R0112	Data Block Segment Selector	8	0008	The data segment is 0x08 (%R).
13	%R0113	Data Block Offset	200	00C8	This offset of 200 places the start of the data block at %R0201. The offset is a zero-based value, but the register tables begin at %R0001. Therefore, the data block starting point is %R0001 + 200 = %R0201.

Continued on Next Page

Word	Register	Parameter	Value (dec)	Value (hex)	Description
Channel Descriptions		The remaining words contain the channel descriptions. In this example six channel descriptions have been defined.			
14	%R0114	Set. Sel. : Length	17921	4601	Channel description 1: The first channel description selects the %I Segment with a length of 1, and an offset of 0. This chooses %I0001 for channel 1.
15	%R0115	Offset	0	0000	
16	%R0116	Seg. Sel. : Length	-253	FF03	Channel description 2: The second channel description selects the NULL Selector with length of 3, and offset of 0. The NULL selector causes channels 2 - 4 to be ignored or "skipped." These channels will always contain a sample value of Zero.
17	%R0117	Offset	0	0000	
18	%R0118	Seg. Sel. : Length	3	0003	Channel description 3: The third channel description selects the Input Module Selector with a length of 3 and offset of 12. The Input Module Selector causes samples to be taken from the input module. This channel description chooses the values in points 13, 14, and 15 of the input module for channels 5 - 7.
19	%R0119	Offset	12	0012	
20	%R0120	Seg. Sel. : Length	18434	4802	Channel description 4: The fourth channel description selects the %Q Segment with a Length of 2, and offset of 8. This chooses %Q0009 and %Q0010 for channels 8 and 9.
21	%R0121	Offset	8	0008	
22	%R0122	Seg. Sel. : Length	8	0008	Channel description 5: The fifth channel description is another Input Module Selector. It has a length of 8, and offset of 0. This causes the values for points 1 to 8 of the input module to be placed in channels 10 - 17.
23	%R0123	Offset	0	0000	
24	%R0124	Seg. Sel. : Length	-249	FF07	Channel description 6: The sixth channel description is another NULL Selector. It has a Length of 7, and offset of 0. This NULL channel description causes channels 18 - 24 to be filled with zeros. This last channel description is required to pad the sample buffer out to the 24 bits specified in the number of channels parameter. Since all 24 channels are configured, no more channel descriptions are needed.
25	%R0125	Offset	0	0000	

Channel Configuration for Above Example

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
U	U	U	U	U	U	U	U	N	N	N	N	N	N	N	C8	C7	C6	C5	C4	C3	C2	C1	%Q 10	%Q 09	C15	C14	C13	N	N	N	%I 01

U = Unused, N = Null, C prefix indicates channel number on configured Input module (for example, C0 = input point 1, C15 = input point 16)

Example Sample Contents

Table 12-2 summarizes the values contained in a single sample based upon the channel descriptions in the sample control block. This is based on the example screen capture shown on the following page. Note in this example that bits 1 – 16 are contained in %R00207 and bits 17 – 24 are part of %R00208.

Table 12-2. Sample Contents for SER Example

Channel Number	Channel Contents	Value
1	%I0001	1
2 - 4	Zeros	000
5	Input Module Point 13	1
6	Input Module Point 14	1
7	Input Module Point 15	1
8	%Q0009	0
9	%Q0010	0
10 - 17	Input Module Points 1 - 8	100100010
18 - 24	Zeros	0000000

Data Block for Control Block Example

Table 12-3 lists the format of the data block resulting from the example control block given on page 12-19. Note that it begins at register 201 as described by the segment offset parameters (Words 12 and 13) in the control block.

Table 12-3. Data Block for SER Control Block Example

Offset	Register	Parameter Description	Value (dec)	Value (hex)
0	%R0201	Current sample offset #	59	003B
1	202	Trigger sample offset #	0	0000
2 - 5	203 – 206	Trigger time (BCD)	0 0 0 0	0000 0000 0000 0000
6 - 768	207 – 975	Sample Buffer	sample data	sample data

Current sample offset is 59, meaning that the 59th sample is the last sample placed in the sample buffer. With 3 bytes per sample, the current offset is actually at 59 * 3 = 177 bytes or the high byte of the 89th register. Since the trigger conditions have not been met, the trigger sample and trigger time are 0 and the output is not set. The sample buffer contains 512 samples where 59 is the newest sample and 60 is the oldest sample.

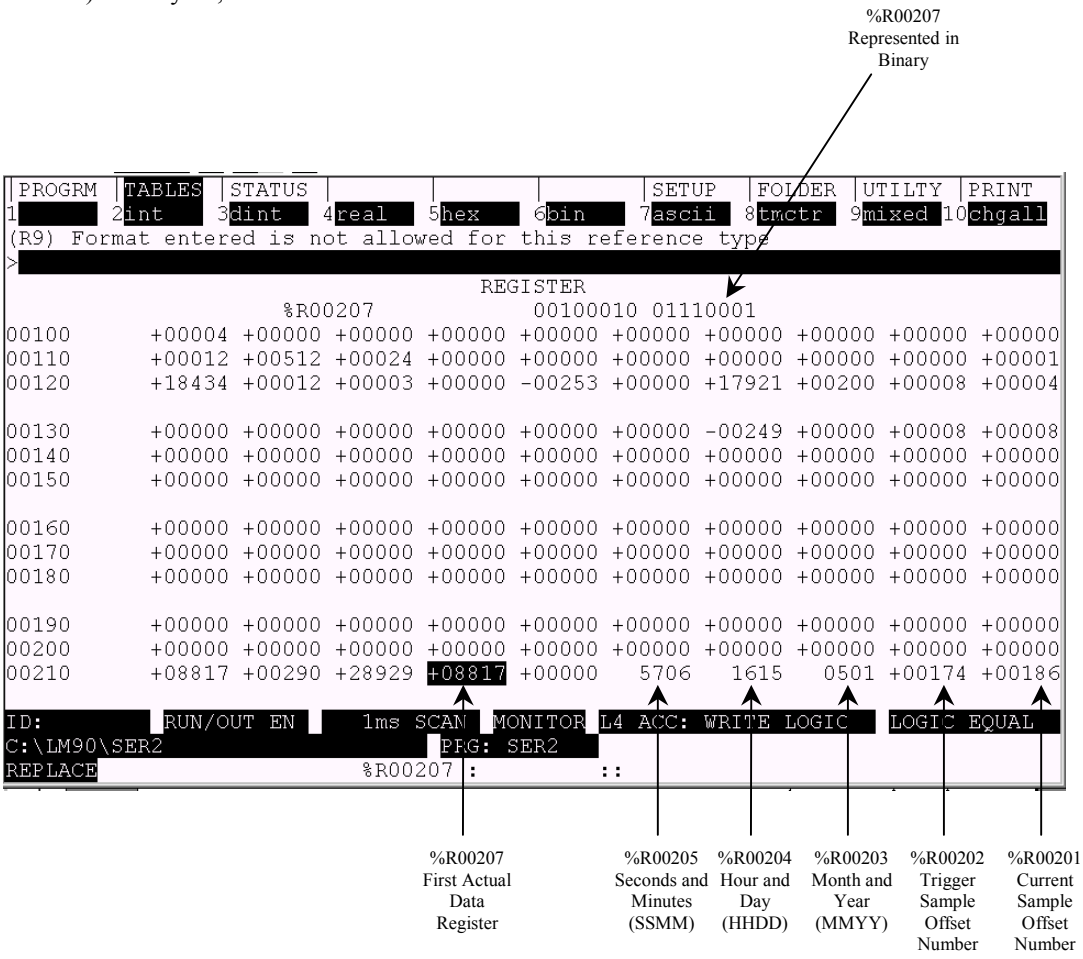
Example Data Capture

Examining the Captured Data

The following screen snap was taken after a trigger. The Control Block starts at %R00100, and the Data Area starts at %R00201. The cursor is positioned on %R00207 as noted near the top and bottom of the screen.

%R00207 is the first register in the data block that actually holds the measured input data. Note that its integer value (-21855) has little meaning in this context; however, by placing your cursor on %R00207, its value is displayed in binary near the top of the screen. Using this binary format, you can determine the states of the bits configured in the Channel Descriptions portion of the control block.

Registers %R00203 through %R00205 give the time and date, in 24-hour format, as 16:06 (and 57 seconds) on May 15, 2001.



END

The END function provides a temporary end of logic. The program executes from the first rung to either the last rung or to the END function, whichever is encountered first.

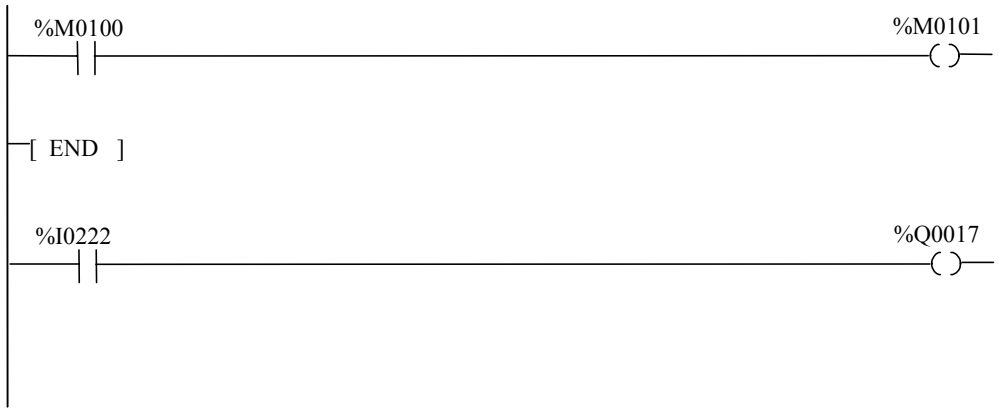
The END function unconditionally terminates program execution. There can be nothing after the end function in the rung. No logic beyond the END function is executed, and control is transferred to the beginning of the program for the next sweep. Note that in rungs past the END marker, inputs will appear to turn on and off, but outputs will not be updated. Although a normal condition, this will appear to be a problem if it isn't apparent that an END marker precedes the affected rungs.

The END function is useful for debugging purposes because it allows you to isolate a section of logic. It does this by preventing any logic that follows it from being executed.

Logicmaster programming software provides, by default, an [END OF PROGRAM LOGIC] marker after the last rung of logic to indicate the end of program execution. This marker is used if no END function is programmed in the logic.

Example

In the following example, the rung containing contact %I0222 and coil %Q0017, and any rungs after it, will not be executed because of the presence of the END instruction.



Note

Placing an END function in SFC logic or in logic called by SFC logic produces an "END Function Executed from SFC Action" fault in Release 7 or later CPUs. (In pre-Release 7 CPUs, it did not work correctly, but no Fault was generated.) For information about this fault, refer to the "System Configuration Mismatch" part of Chapter 3, Section 2.

MCRN/MCR

Overview of MCR and MCRN

A Master Control Relay (MCR/MCRN) function must be used with a corresponding End Master Control Relay (ENDMCR/ENDMCRN) function. Both functions must have the same name. The MCR/MCRN must have an enable contact between it and the power rail. All rungs between an enabled MCR/MCRN and its corresponding ENDMCR/ENDMCRN function are executed without power flow to coils. The ENDMCR/ENDMCRN function associated with the MCR/MCRN causes normal program execution to resume. Unlike the JUMP instruction, an MCR/MCRN can only occur in the forward direction. An ENDMCR/MCRN instruction must appear later in a program than its corresponding MCR/MCRN instruction.

The following controls are imposed on logic controlled by an enabled MCR/MCRN:

- Timers do not increment or decrement. Any TMR type timer is reset (accumulator is set to zero). For an ONDTR timer, the accumulator is "frozen" at the value that was current when the MCR/MCRN was enabled.
- Power flow does not occur for any instruction. Normal outputs are off; negated outputs are on.
- Instructions do not update their outputs. For example, an ADD instruction will not produce a current sum in its Q output register, a Move will not copy its current input value to its output, a Shift Register will not shift data, etc. The values in these output registers will be frozen at the values that were present when the MCR/MCRN was enabled.

Note

When an MCR/MCRN is energized, the logic it controls is evaluated and contact status is displayed, but no outputs are energized. If you are not aware that an MCR/MCRN is controlling the logic being observed, this might appear to be a faulty condition. To indicate that a range of ladder logic is under MCR/MCRN control, Logicmaster displays a double power rail on the ladder logic screen. This double power rail appears regardless of whether or not the MCR/MCRN is enabled.

Logicmaster 90-30/20/Micro software supports two forms of the Master Control Relay function, an older, non-nested (MCR) and a newer, nested form (MCRN).

CPU Compatibility

CPU Type	Supported Form
CPU311 – CPU341, Release 1	Use only the non-nested form (MCR)
CPU311 – CPU341, Release 2 and later	Use only the nested form (MCRN)
35x, 36x, and 37x series CPUs	Use only the nested form (MCRN)

Possible MCRN Compatibility Problem

When converting a CPU340 or CPU341 program to run in a 35x/36x/37x series CPU, it is possible to see a "Feature not Supported" error ("Nesting Levels Exceeded") from Logicmaster 90. This would occur when the converted program is stored to a 35x/36x/37x CPU if more than eight levels of MCRN nesting is used in the original program.

The MCRN instructions are actually function block instructions in the CPU340/341, which means they are executed in CPU firmware and not executed by the embedded Boolean Coprocessor (BCP). The nesting limit for the function block was set to 256. This limit is many more levels than you would generally use. When the 35x/36x/37x CPU series was designed, the MCRN instructions were moved to the BCP to improve CPU performance (function block instructions execute slower than BCP counterparts). At that time a tradeoff of nesting levels and performance was made and the BCP3 used in 35x/36x/37x CPUs implemented eight levels of nesting, which are normally more than users require. So, Logicmaster 90 enforces eight levels of nesting when the program conversion is performed, and if there are more than eight levels used, a "Nesting Levels Exceeded" message is issued.

Therefore, if you have more than eight MCR nesting levels in a CPU340/341 program, it will require a modification to work in a 35x/36x/37x CPU. You might consider using Jump statements instead.

Nesting an MCRN

An MCRN function can be placed anywhere within a program, as long as it is properly nested with respect to other MCRNs, and does not occur in the range of any non-nested MCR or non-nested JUMP.

If an MCRN/ENDMCRN pair is nested within another MCRN/ENDMCRN pair, it must be contained completely within the other pair. Up to eight levels of nesting are allowed. For an example, see page 12-28.

There can be multiple MCRN functions corresponding to a single ENDMCRN (except for the 35x/36x/37x series CPUs as noted below). Each MCRN as well as the ENDMCRN must have the same name. This is analogous to the nested JUMP, where you can have multiple JUMPs to the same LABEL. For a comparison of the JUMP function and the MCR function, refer to the "Differences Between MCRs and Jumps" section below.

Note

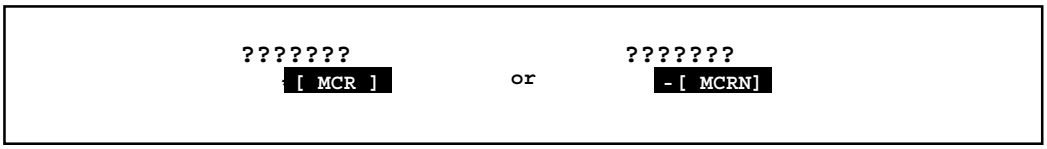
Use only one MCRN for each ENDMCRN with 35x, 36x and 37x series CPUs.

MCR Operation

There can be only one MCR instruction for each ENDMCR instruction. The range for non-nested MCRs and ENDMCRs cannot overlap or contain the range of any other MCR/ENDMCR pair or any JUMP/LABEL pair of instructions. Non-nested MCRs cannot be within the scope of any JUMP/LABEL pair.

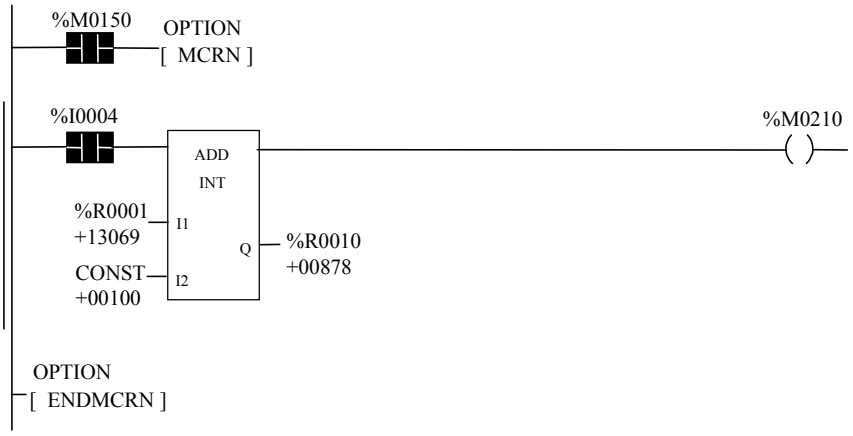
Parameters

Both forms of the MCR function have the same parameters. They both have an enable Boolean input EN and a name that identifies the MCR. This name is used again with an ENDMCR instruction. Neither the MCR nor the MCRN function has any outputs; there can be nothing after an MCR in a rung.

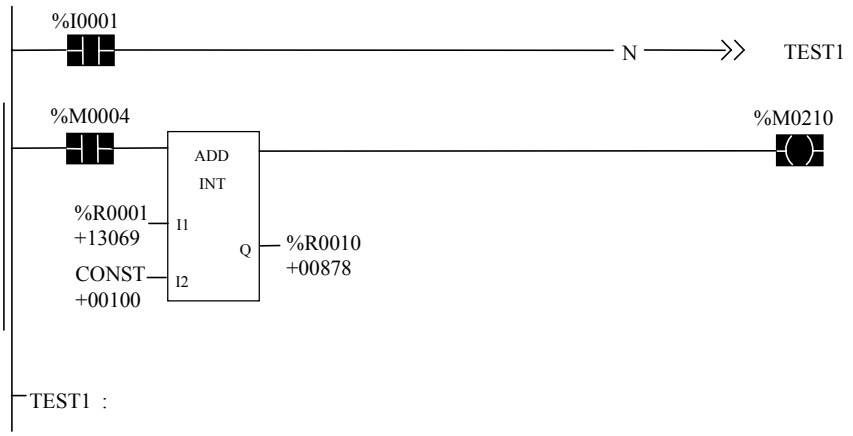


Differences Between MCR/MCRN and JUMP

With an MCR function, function blocks within the scope of the MCRN are evaluated ***without power flow***, and coils ***are not energized***. In the following example, when %M0150 is ON, the MCRN is enabled. When the MCRN is enabled, even if %I004 is ON, the ADD function block is evaluated ***without*** power flow (i.e., it does not add 100 to %R0001), and %M0210 does not receive power flow. Status of contacts such as %I0004 and values in registers used on inputs, such as %R0001, will update on the Logicmaster screen, but registers on outputs under control of the MCRN, such as %R0010, will be frozen at their current values when the MCRN is enabled.



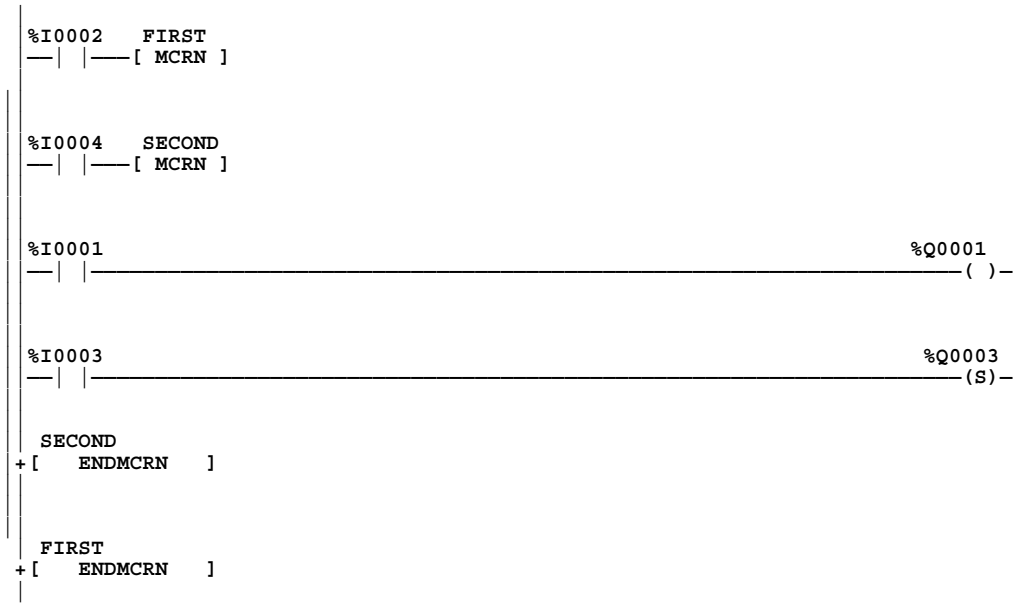
With a JUMP function, any function blocks between the JUMP and the LABEL ***are not*** evaluated, and coils ***are not affected***. In the following example, when %I0001 is ON, the JUMP named TEST1 is enabled. Since the logic between the JUMP and the LABEL is skipped, %M0210 is unaffected (i.e., if it was ON, it remains ON; if it was OFF, it remains OFF). Status of contacts such as %M0004 and values in registers used on inputs, such as %R0001, will update on the Logicmaster screen, but registers on outputs under control of the JUMP, such as %R0010, will be frozen at their current values when the JUMP is enabled.



Example 1

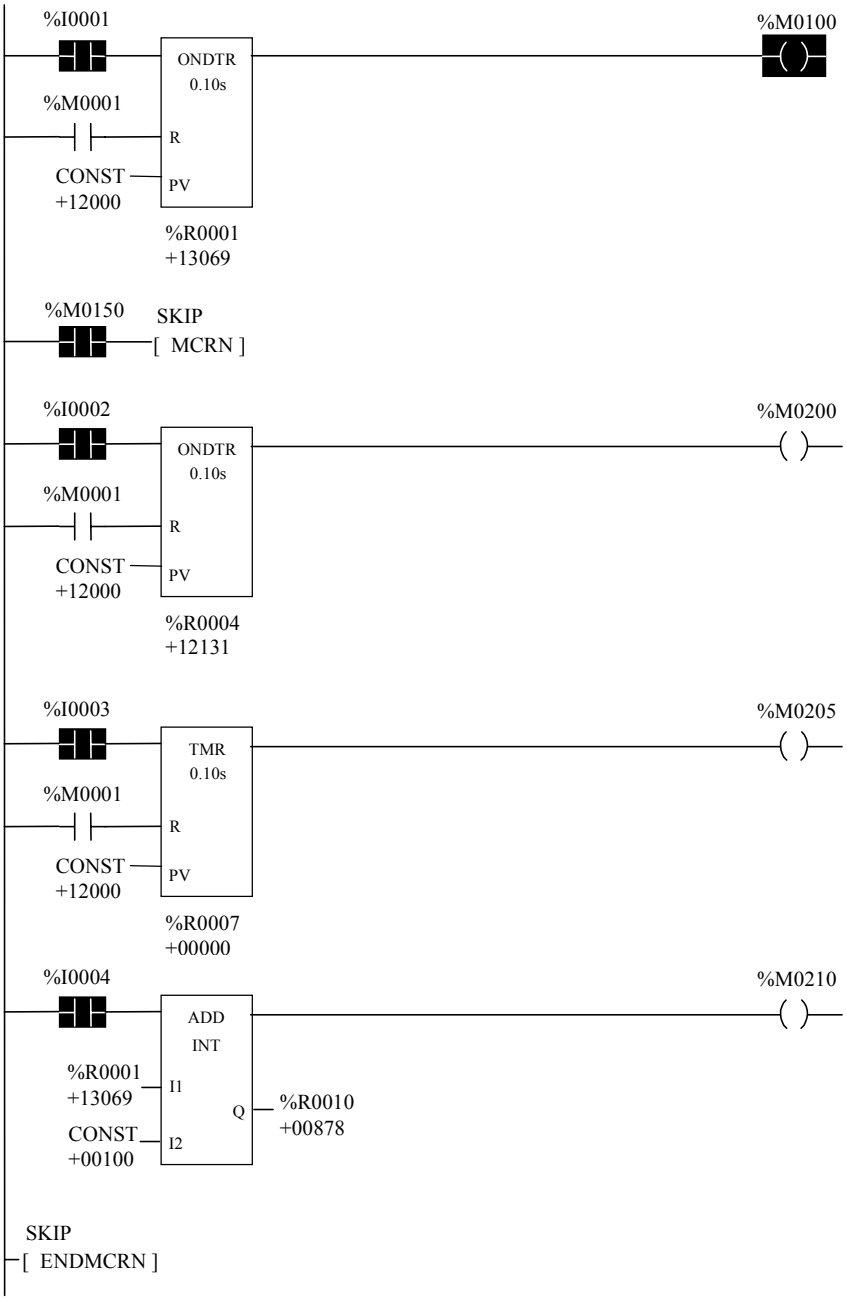
The following example shows an MCRN named “Second” nested inside the MCRN named “First.” Whenever %I0002 allows power flow into the MCRN function, program execution will continue without power flow to the coils until the associated ENDMCRN is reached. If %I0001 and %I0003 are ON, %Q0001 is turned OFF and %Q0003 remains ON.

To aid in troubleshooting ladder programs, a double power rail identifies logic that is within the control range of an MCR.



Example 2

In the following example, the first rung is functioning normally. However, the MCRN named SKIP is controlling the rest of the rungs, which have a double power bar to indicate this. In the first rung controlled by the MCRN, the ONDTR timer's accumulated value (%R0004) is frozen, and even though it reached its preset value, its output (%M0200) is not energized. In the following rung, the TMR has been reset by the MCRN. Its accumulated value (%R0007) is held at zero and its output (%M02025) is not energized. In the next rung, the ADD instruction's output is frozen (its output at %R0010 is not the sum of its inputs) and its power flow coil (%M0210) is not energized. Note, however, that the status of contacts and values of input registers (such as %R0001 on the ADD instruction I1 input) are updated on-screen within the MCRN control area.

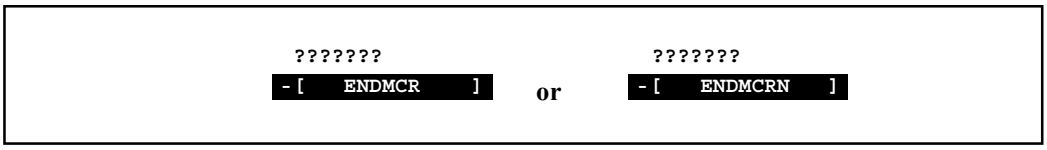


ENDMCRN/ENDMCR

Use the End Master Control Relay ENDMCR/ENDMCRN function to resume normal program execution after an MCR/MCRN 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.

Logicmaster 90-30/20/Micro software supports two forms of the ENDMCR function, a non-nested and a nested form. The non-nested form, ENDMCR, must be used with the non-nested MCR function, MCR. The nested form, ENDMCR, must be used with the nested MCR function, MCRN.

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.



Example

In the following examples, an ENDMCR instruction is programmed to terminate the MCR named "CLEAR."

Example of a non-nested ENDMCR

```
  CLEAR
-[   ENDMCR    ]
```

Example of a nested ENDMCR:

```
  CLEAR
-[   ENDMCRN   ]
```

JUMP

Use the JUMP instruction to cause a portion of the program logic to be bypassed. Program execution will continue at the LABEL specified. When the JUMP is active, all coils within its scope are left at their previous states. This includes coils associated with timers, counters, latches, and relays.

Logicmaster 90-30/20/Micro software supports two forms of the JUMP instruction, a non-nested and a nested form. The non-nested form has been available since Release 1 firmware for the CPU311-CPU341 CPUs, and has the form ————————>>LABEL01, where LABEL01 is the name of the corresponding non-nested LABEL instruction.

For non-nested JUMPs, there can be only a single JUMP instruction for each LABEL instruction. The JUMP can be either a forward or a backward JUMP.

The range for non-nested JUMPs and LABELs cannot overlap the range of any other JUMP/LABEL pair or any MCR/ENDMCR pair of instructions. Non-nested JUMPs and their corresponding LABELs cannot be within the scope of any other JUMP/LABEL pair or any MCR/ENDMCR pair. In addition, an MCR/ENDMCR pair or another JUMP/LABEL pair cannot be within the scope of a non-nested JUMP/LABEL pair.

Note

The non-nested form of the JUMP instruction is the only JUMP instruction that can be used in a Release 1 Series 90-30 PLC. The nested JUMP function can be used (and is suggested for use) for all new applications.

Also, please note that the 35x/36x/37x series CPUs support only nested jumps.

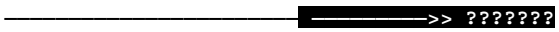
The nested form of the JUMP instruction has the form ———N——>>LABEL01, where LABEL01 is the name of the JUMP and its corresponding nested LABEL instruction. The nested JUMP is available in Release 2 and later releases of Logicmaster 90-30/20/Micro software and PLC firmware.

A nested JUMP instruction can be placed anywhere within a program, as long as it does not occur in the range of any non-nested MCR or non-nested JUMP.

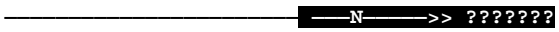
There can be multiple nested JUMP instructions corresponding to a single nested LABEL. Nested JUMPs can be either forward or backward JUMPs.

Both forms of the JUMP instruction are 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.

Non-nested JUMP:



Nested JUMP:

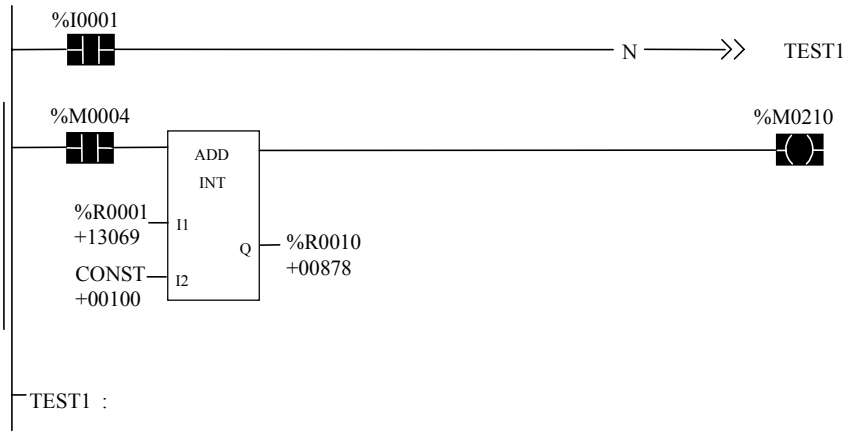


Caution

To avoid creating an endless loop with backward JUMP instructions, a backward JUMP must contain a way to make it conditional.

Examples

In the following example, whenever contact %I0001 turns on, the JUMP named TEST1 is enabled, and power flow is jumped ahead to the TEST1 LABEL. Since the logic between the JUMP and the LABEL is skipped, %M0210 is unaffected (i.e., if it was ON, it remains ON; if it was OFF, it remains OFF). Status of contacts such as %M0004 and values in registers used on inputs, such as %R0001, will update on the Logicmaster screen, but registers on outputs under control of the JUMP, such as %R0010, will be frozen at their current values when the JUMP is enabled. Note the use of the double power rail in the section of logic located between the JUMP and its LABEL.



LABEL

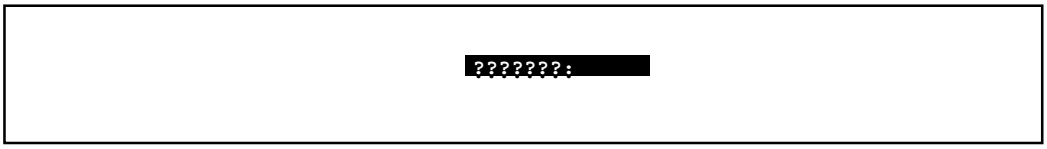
The LABEL instruction functions as the target destination of a JUMP. Use the LABEL instruction to resume normal program execution after a JUMP instruction.

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.

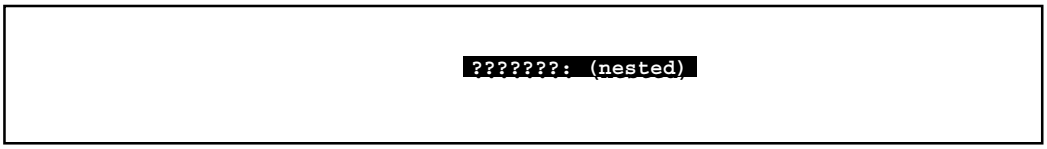
Logicmaster 90-30/20/Micro software supports two forms of the LABEL function, a non-nested and a nested form. For example, the non-nested form, LABEL01 :, must be used with the non-nested JUMP function, ————————>>LABEL01; the nested form, LABEL01 : (nested), must be used with the nested JUMP function, ————N——>>LABEL01.

The LABEL instruction has no inputs and no outputs. Also, there can be nothing either before or after a LABEL in a rung.

Non-nested LABEL:

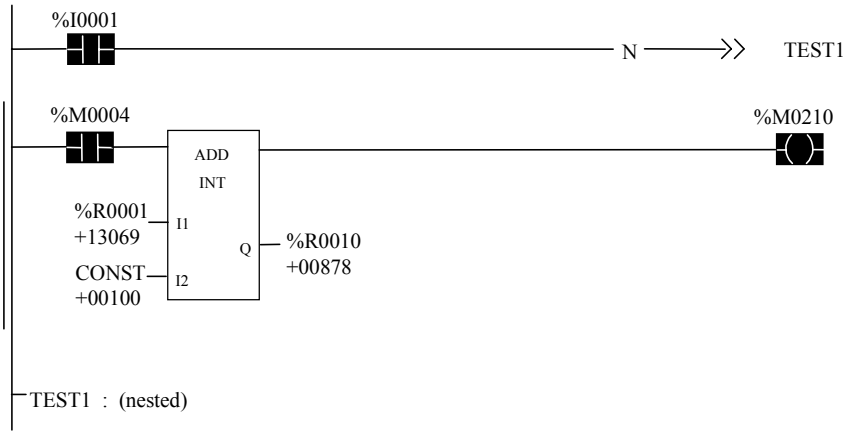


Nested LABEL:



Example

In the following example, when JUMP TEST1 is enabled, the scan skips ahead to the TEST1 : (nested) LABEL, which means that the rung in-between the JUMP and LABEL is not scanned.



COMMENT

Comments are useful for adding explanations, notes, revision level information, etc. to your ladder program. Use of comments is highly recommended because they provide valuable information to those who may have to troubleshoot or update the system in the future. Also, since human memories are imperfect, comments are valuable references for even the creator of the ladder program.

Note

To conserve PLC memory, annotations (comments, nicknames, and descriptions) are not written to the PLC. Therefore, to view these annotations, you must have a copy of the original program folder (which includes the annotations) on your computer. Then, when you connect your computer to the PLC, the links to the annotations will automatically be made by your programming software.

Creating a Standard Comment

A comment can have up to 2048 characters of text. In Logicmaster, it is represented in the ladder logic like this:

```
(*  COMMENT  *)
```

Creating a Comment

1. Create a new rung. A COMMENT rung cannot have any other logic besides the COMMENT instruction.
2. Insert the COMMENT, which is found in the Control group of instructions.
3. Accept the rung by pressing the Escape key.
4. Move the cursor over the (* COMMENT *) instruction just created and press the Zoom key (F10) to enter the comment editor screen.
5. Type in your comment text. Note that the lines do not automatically wrap in the comment editor. You must press the Enter key at the end of a line to begin typing on the next line.
6. When finished, press Escape key to exit the comment editor and save the comment.

Once created, COMMENT text can be read or edited by moving the cursor to (* COMMENT *) and selecting Zoom (F10). Rung Comments can also be printed from Logicmaster's Print menu.

Creating a Long Comment for use in Logicmaster Printouts

In Logicmaster longer text can be included in printouts using an annotation text file:

1. Create the comment (see previous section for comment creation details):
   A. Enter comment text to the point where the text from the other file should begin.
   B. On a new line, enter **\I** (or **\i**), the drive letter followed by a colon, a backslash, the subdirectory or folder, a backslash, and the file name, as shown in this example:

      ```
      \I d:\text\commnt1
      ```

      (Drive designation is not necessary if the file is on the same drive as the program folder.)
   C. Press Escape to exit the comment editor and save the comment text.
2. Open a text processor and create a text file.
3. Save the text file in a .txt format, giving it the file name entered in the comment, and saving it on the drive and in the path specified in the comment.

SVCREQ

The Service Request instruction is a general purpose instruction that can perform a wide variety of special instructions (services) that are not available as individual function blocks. Use the Service Request (SVCREQ) function to request one of the following special PLC services:

Table 12-4. Service Request Functions

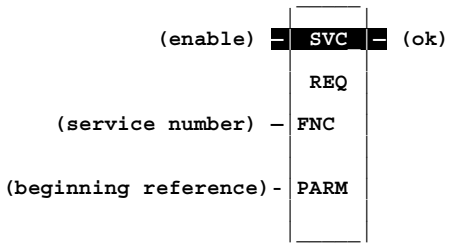
Function	Description
1	Change/Read Constant Sweep Timer.
2	Read Window Values.
3	Change Programmer Communications Window Mode and Timer Value.
4	Change System Comm. Window Mode and Timer Value.
6	Change/Read Checksum Task State and Number of Words to Checksum.
7	Change/Read Time-of-Day Clock.
8	Reset Watchdog Timer.
9	Read Sweep Time from Beginning of Sweep.
10	Read Folder Name.
11	Read PLC ID.
12	Read PLC Run State.
13	Shut Down the PLC.
14	Clear Fault Tables.
15	Read Last-Logged Fault Table Entry.
16	Read Elapsed Time Clock.
18	Read I/O Override Status.
23	Read Master Checksum.
24	Reset Smart Module
26/30	Interrogate I/O.
29	Read Elapsed Power Down Time.
45	Skip Next Output and Input Scan. (Suspend I/O.)
46	Access Fast Backplane Status.
48	Reboot After Fatal Fault Auto Reset
49	Auto Reset Statistics

SVC REQ Overview

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 can represent any type of word memory (%R, %AI, or %AQ). This reference 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 can be used both as inputs for the function and as the location where data is output after the function executes. Therefore, data returned by the function is accessed at the same location specified for PARM.



Parameters

Parameter	Description
enable	When enable is on, the service request is performed.
FNC	Each type of Service Request has a unique function number, which must be programmed at the FNC input. FNC may contain either a constant or a reference address that contains the function number of 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.

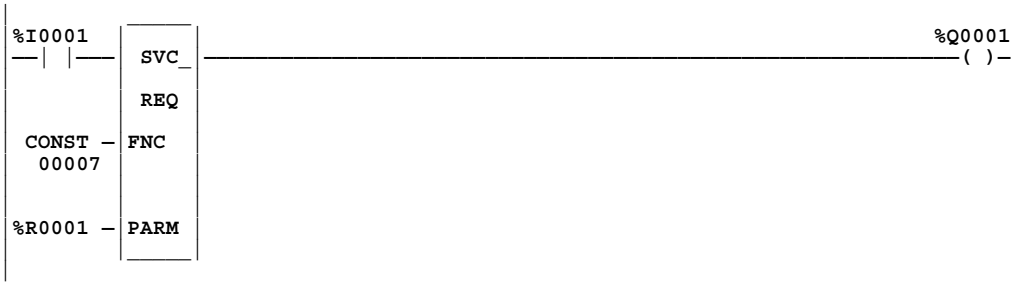
Valid Memory Types

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
FNC		•	•	•	•		•	•	•	•	•	
PARM		•	•	•	•		•	•	•	•		
ok	•											•

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

Example

In the following example, when enable contact %I0001 is ON, SVCREQ function number 7, specified at input FNC, is performed. The function's parameter block starts at %R0001 (specified at PARM). Output coil %Q0001 is set ON if the operation succeeds.



SVCREQ #1: Change/Read Constant Sweep Timer

Beginning with 90-30 CPU Release 8.0, 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.

Note

Of the CPUs discussed in this manual, Service Request 1 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

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 **without** 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 8 and higher CPUs will provide the return values 0 for Normal Sweep, 1 for Constant Sweep. 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.

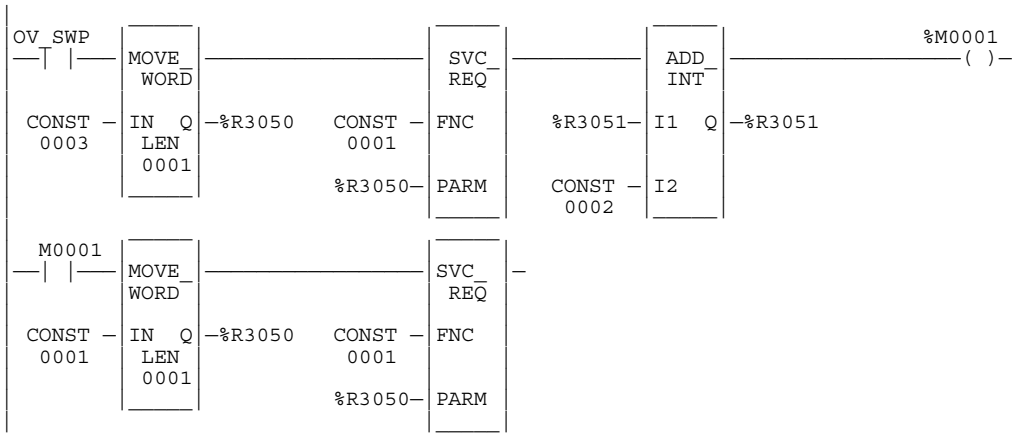
After the function executes, the function returns the timer state and value in the same parameter block references:

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 %R3050. Because the MOVE and ADD functions require three horizontal contact positions, the example logic uses discrete internal coil %M0001 as a temporary location to hold the successful result of the first rung line. On any sweep in which OV_SWP is not set, %M0001 is turned off.



SVCREQ #2: Read Window Values

Use SVCREQ function #2 to obtain the current window mode time values for the programmer communications window and the system communications window.

Note

Of the CPUs discussed in this manual, Service Request 2 is supported only by 90-30 CPUs, beginning with Release 8.0.

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 between the two 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:

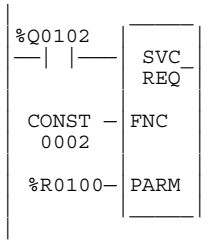
	High Byte	Low Byte	
Programmer Window	Mode	Value in ms	address
System Communications Window	Mode	Value in ms	address + 1
Reserved*	*See Note	*See Note	address + 2

* Note. The address + 2 word is reserved for use by the system. All zeros will be returned here.

All parameters are output parameters. It is not necessary to enter values in the parameter block to program this function. Output values for both window are given in milliseconds.

Example

In the following example, when enabling output %Q0102 is set, the PLC operating system places the current time values of the three windows in the parameter block starting at location %R0100. Additional examples showing the Read Window Values function are included in the next three SYS REQ function descriptions.



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.

Note

Of the CPUs discussed in this manual, Service Request 3 is supported only by 90-30 CPUs, beginning with Release 8.0.

The SVCREQ function #3 will pass power flow to the right unless a mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.

The parameter block has a length of one word.

To disable the programmer window, enter SVCREQ function #3 with this parameter block:

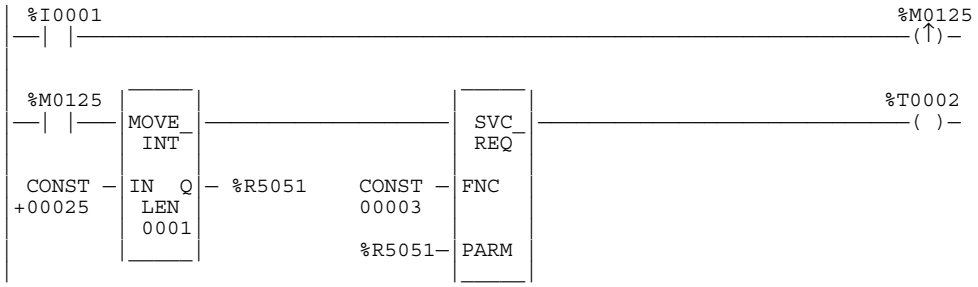
High Byte	Low Byte	
0	0	address

To enable the programmer window, enter SVCREQ function #3 with this parameter block:

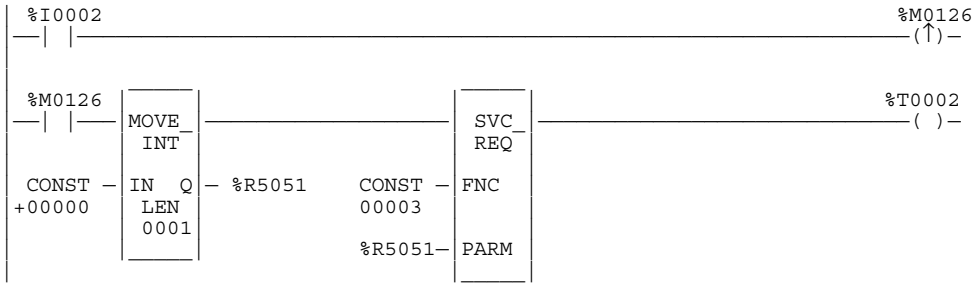
High Byte	Low Byte	
Mode	Value from 1 to 255 ms	address

Example

In the following example, when %M0125 transitions on, the programmer communications window is enabled and assigned a value of 25 ms. The parameter block is in memory location %R5051.



To disable the programmer communications window, use Service Request 3 to assign a value of zero (0). In this example, when %M0126 transitions on, the programmer communications window is enabled and assigned a value of 0 ms. The parameter block is in memory location %R5051.



SVCREQ #4: Change System Comm 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.

Note

Of the CPUs discussed in this manual, Service Request 4 is supported only by 90-30 CPUs, beginning with Release 8.0.

The SVCREQ function #4 will pass power flow to the right unless a mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) 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:

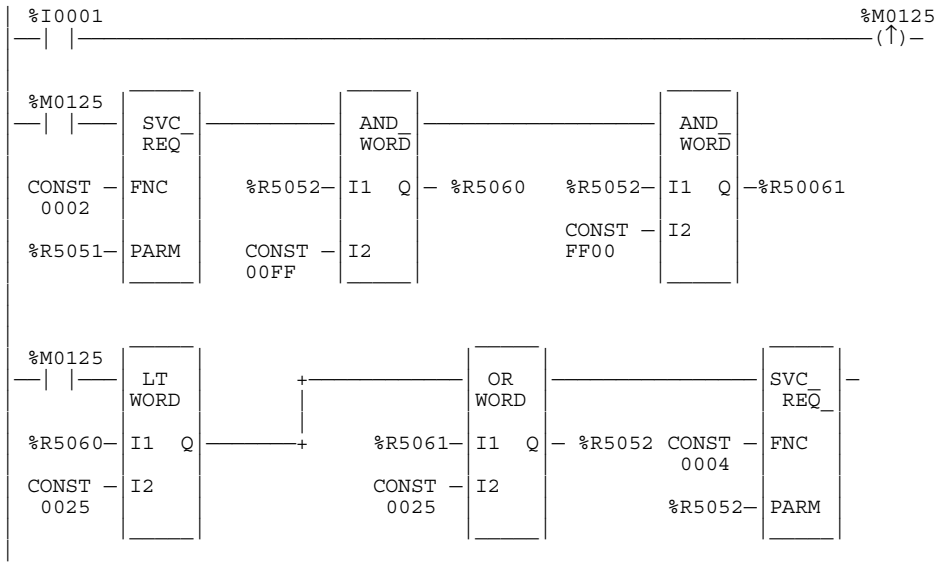
High Byte	Low Byte	
0	0	address

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 %M0125 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 %R5051. 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 %R5052.



SVCREQ #6: Change/Read Number of Words to Checksum

Use the SVCREQ function with function number 6 in order to:

- Read the current word count.
- Set a new word count.

Successful execution will occur, unless some number other than 0 or 1 is entered as the requested operation (see below).

For the Checksum Task functions, the parameter block has a length of 2 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 (0 – 32)	address + 1

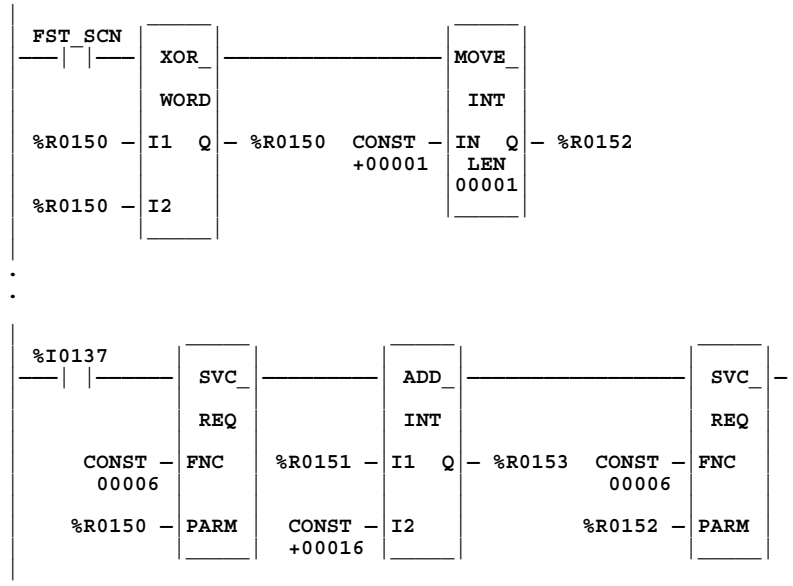
Entering 1 causes the PLC to adjust the number of words to be checksummed to the value given in the second word of the parameter block. For any Series 90-30 CPU, the second word value can be from 0 to 32. If the value is outside this range, an error will be generated. For the Series 90-20 CPU211, the value can be either 0 or 4.

Note

This Service Request is not available on Micro PLCs.

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 %I0137 turns 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_INT 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.



The example parameter blocks are located at address %R0150. They have the following content:

0 = read current count	%R0150
Hold current count	%R0151

1 = set current count	%R0152
Hold new count for set	%R0153

SVCREQ #7: Change/Read Time-of-Day Clock

Use the SVCREQ function with function number 7 to read and set the time-of-day clock in the PLC.

Note

This function is available only in 331 or higher 90-30 CPUs and on the 28-point Series 90 Micro PLC CPUs (that is, IC693UDR005, IC693UAA007, and IC693UDR010) and the 23-point Series 90 Micro PLC CPUs (IC693UAL006).

Successful execution will occur unless:

1. Some number other than 0 or 1 is entered as the requested operation (see below).
2. An invalid data format is specified.
3. The data provided is not in the expected format.
4. An invalid date is entered, such as 02/29/01, which incorrectly specifies a leap year day in the year 2001 (2001 is not a leap year).

For the date/time functions, the length of the parameter block depends on the data format. BCD format requires 6 words; packed ASCII requires 12 words.

0 = read time and date 1 = set time and date	address
1 = BCD format 3 = packed ASCII format	address + 1
data	address + 2 to end

In word 1, specify whether the function should read or change the values.

```
0    =    read
1    =    change
```

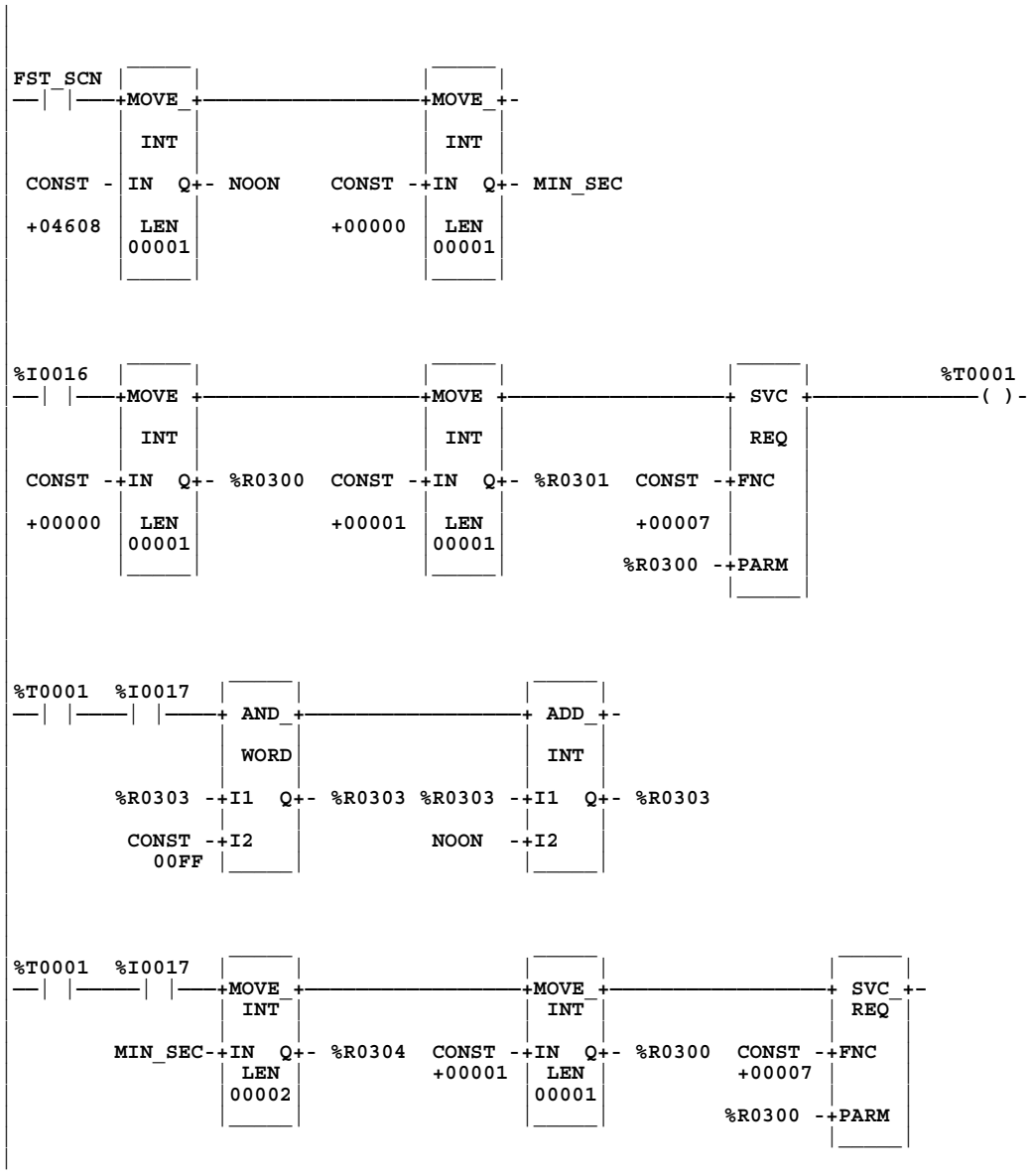
In word 2, specify a data format:

```
1    =    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.

Example

In the following example, when called for by previous logic, 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 %R0300. 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.



Parameter Block Contents

Parameter block contents for the different data formats are shown on the following pages. For both data formats:

- Hours are stored in 24-hour format.
- 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 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).

High Byte		Low Byte		Example output parameter block: Read Date and Time in BCD format (Sun., July 3, 1988, at 2:45:30 p.m.)	
1 = change	or	0 = read	address	0	
1			address + 1	1	
month		year	address + 2	07	88
hours		day of month	address + 3	14	03
seconds		minutes	address + 4	30	45
(null)		day of week	address + 5	00	01

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.

High Byte	Low Byte		Example output parameter block: Read Date and Time in Packed ASCII Format (Mon, Oct. 2, 1989 at 23:13:00)	
1 = change or	0 = read	address	0	
3		address + 1	3	
year	year	address + 2	39	38
month	(space)	address + 3	31	20
(space)	month	address + 4	20	30
day of month	day of month	address + 5	32	30
hours	(space)	address + 6	32	20
:	hours	address + 7	3A	33
minutes	minutes	address + 8	33	31
seconds	:	address + 9	30	3A
(space)	seconds	address + 10	20	30
day of week	day of week	address + 11	32	30

SVCREQ #8: Reset Watchdog Timer

Use SVCREQ function #8 to reset the watchdog timer during the sweep.

Note

Of the CPUs discussed in this manual, Service Request 8 is supported only by 90-30 CPUs, beginning with Release 8.0.

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).

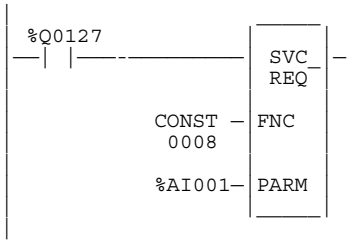
Caution

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 %Q0127 turns ON, the watchdog timer is reset.



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 is in 16-bit Word format.

Note

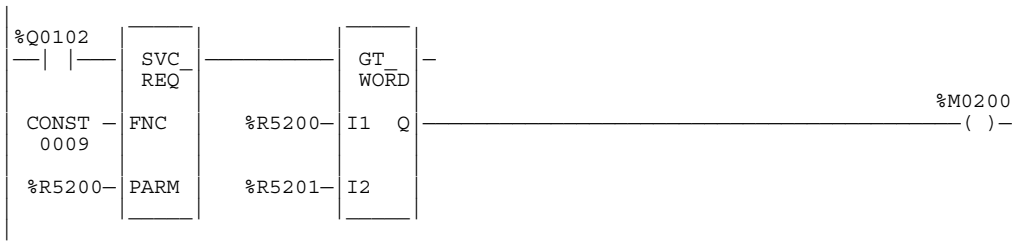
Of the CPUs discussed in this manual, Service Request 9 is supported only by 90-30 CPUs, beginning with Release 8.0.

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 %R5200. If it is greater than the value in %R5201, internal coil %M0200 is turned on.



SVCREQ #10: Read Folder Name

Use SVCREQ function #10 to read the name of the currently-executing folder.

Note

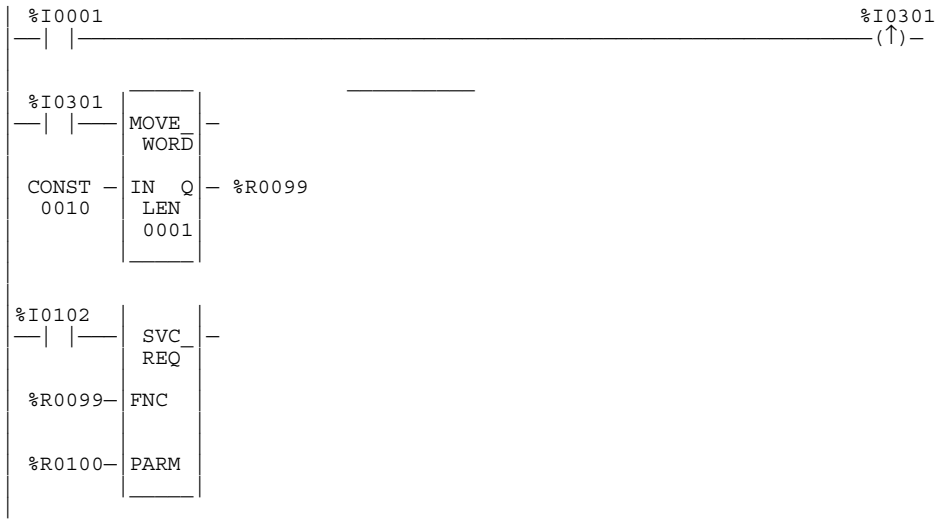
Of the CPUs discussed in this manual, Service Request 10 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

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 contact %I0301 transitions ON, register location %R0099 is loaded with the value 10, which is the function code for the Read Folder Name function. In the following rung, when %I0102 is ON, the Service Request reads the folder name and stores it in the four-word block of memory starting at %R0100 (specified at PARM).



SVCREQ #11: Read PLC ID

Use SVCREQ function #11 to read the name of the Series 90 PLC executing the program.

Note

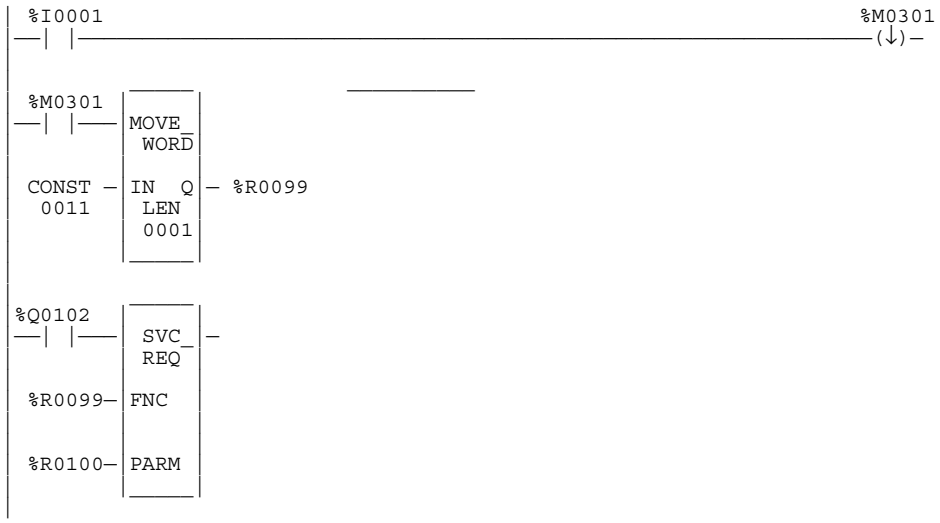
Of the CPUs discussed in this manual, Service Request 11 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

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 contact %I0001 transitions OFF, register location %R0099 is loaded with the value 11, which is the function code for the Read PLC ID function. . In the following rung, when %Q0102 is ON, the Service Request reads the PLC ID and stores it in the four-word block of memory starting at %R0100 (specified at PARM).



SVCREQ #12: Read PLC Run State

Use SVCREQ function #12 to read the current RUN state of the PLC CPU.

Note

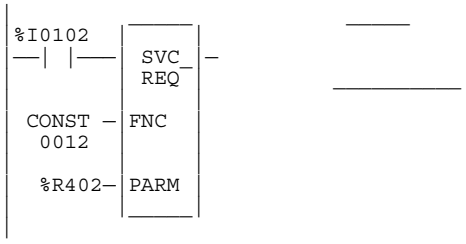
Of the CPUs discussed in this manual, Service Request 12 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The parameter block is an output parameter block only; it has a length of one word. There are only two valid results obtainable from the execution of this Service Request:

1 = run/disabled 2 = run/enabled	address

Example

In the following example, when %I0102 turns ON, the Service Request reads the PLC run state and places the result in memory address %R402. If the PLC is in Run/Disabled mode, %R402 will contain a value of 1. If the PLC is in Run/Enabled mode, %R402 will contain a value of 2.



SVCREQ #13: Shut Down (Stop) PLC

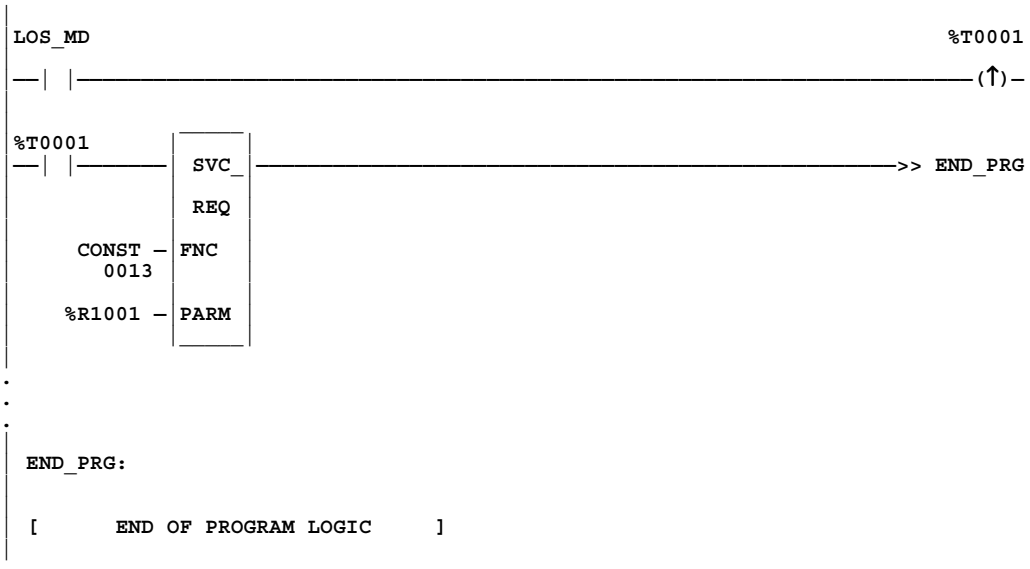
Use SVCREQ function #13 in order to stop the PLC ***at the end of the next sweep***. All outputs will go to their designated default states at the beginning of the next PLC sweep. An informational fault is placed in the PLC fault table, noting that a "SHUT DOWN PLC" function block was executed. The I/O scan will continue as configured.

This function has no parameter block.

Example

In the following example, when a "Loss of I/O Module" 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 Shut Down 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. Once the PLC receives this STOP command from the Service Request, it will execute one more sweep and then stop (see NOTE below).



Note

To ensure that the %S0002 LST_SCN contact will operate correctly, the PLC will execute one additional sweep after the sweep in which the SVCREQ function #13 was executed.

SVCREQ #14: Clear Fault Tables

Use SVCREQ function #14 in order to clear either the PLC fault table or the 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).

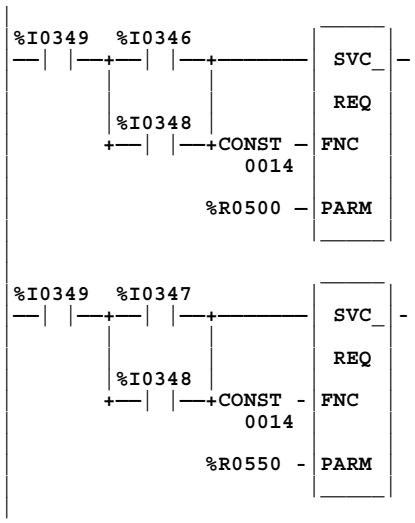
For this function, the parameter block has a length of 1 word. It is an input parameter block only.

0 = clear PLC fault table. 1 = clear I/O fault table.	address

Example

In the following example, when contacts %I0346 and %I0349 are both on, the PLC fault table is cleared. When contacts %I0347and %I0349 are both on, the I/O fault table is cleared. When contacts %I0348 and %I0349 are both on, both fault tables are cleared.

The parameter block for the PLC fault table is located at %R0500, and for the I/O fault table the parameter block is located at %R0550. Both parameter blocks are set up elsewhere in the program (they both must be at logic 1 in order to clear their respective tables).



SVCREQ #15: Read Last-Logged Fault Table Entry

Use SVCREQ function #15 in order to read the last entry logged in either the PLC fault table or the 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), or the fault table is empty. (For additional information on fault table entries, refer to chapter 3, "Fault Explanations and Correction.")

For this function, the parameter block has a length of 22 words. The input parameter block has this format:

0 = Read PLC fault table. 1 = Read I/O fault table.	address

The format for the output parameter block depends on whether the function reads data from the PLC fault table or the I/O fault table.

PLC Fault Table Output Format

Low Byte	High Byte	
0		
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
time stamp		address + 19
		address + 20
		address + 21

I/O Fault Table Output Format

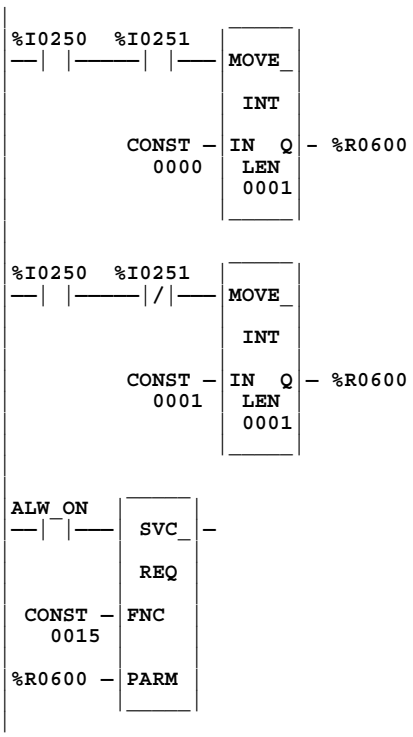
Low Byte	High Byte
1	
long/short	
reference address	
I/O fault address	
fault group and action	
fault category	fault type
fault description	
fault specific data	
time stamp	

In the first byte of word address + 1, the Long/Short indicator defines the quantity of fault specific data present in the fault entry. It can be:

```
PLC Fault Table:   00 = -8 bytes (short)
                   01 = 24 bytes (long)
I/O Fault Table:   02 = –5 bytes (short)
                   03 = 21 bytes (long)
```

Example 1

In the following example, when input %I0251 is on and input %I0250 is on, the last entry in the PLC fault table is read into the parameter block. When input %I0251 is off and input %I0250 is on, the last entry in the I/O fault table is read into the parameter block. The parameter block is located at location %R0600.



Example 2

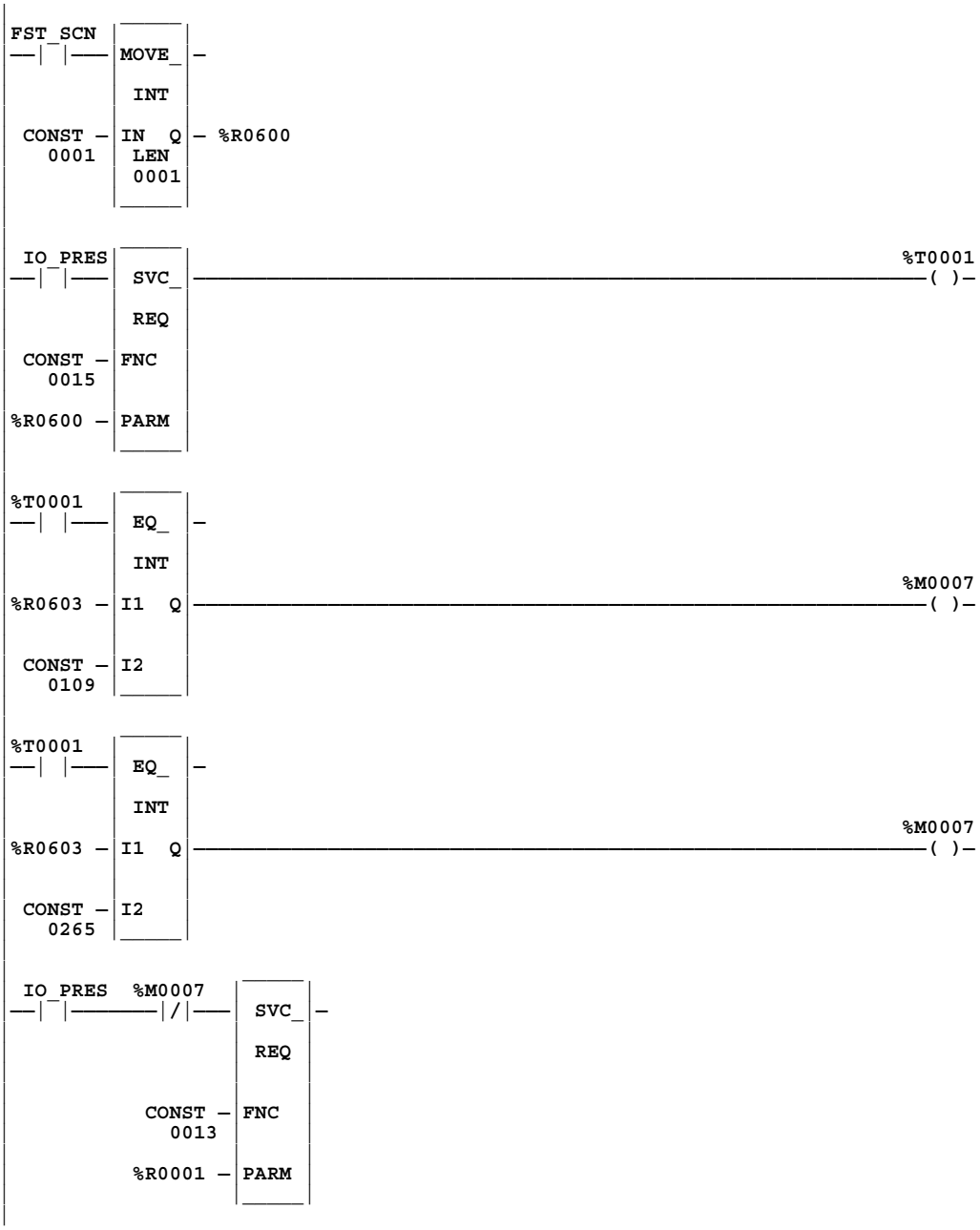
In the next example, the PLC is shut down when any fault occurs on an I/O module except when the fault occurs on modules in rack 0, slot 9 and in rack 1, slot 9. If faults occur on these two modules, the system remains running. The parameter for “table type” is set up on the first sweep. The contact IO_PRES, when set, indicates that the I/O fault table contains an entry. The PLC CPU sets the normally open contact in the next sweep after the fault logic places a fault in the table. If faults are placed in the table in two consecutive sweeps, the normally open contact is set for two consecutive sweeps.

The example uses a parameter block located at %R0600. After the SVCREQ function executes, the fourth word of the parameter block contains the rack and slot location of the I/O module that faulted:

1		%R0600
long/short		%R0601
reference address		%R0602
rack number	slot number	%R0603
I/O bus no.	bus address	%R0604
point address		%R0605

fault data

In the program, the EQ_INT blocks compare the rack/slot address in the table to hexadecimal constants. The internal coil %M0007 is turned on when the rack/slot where the fault occurred meets the criteria specified above. If coil %M0007 is on, its normally closed contact is off, preventing the shutdown. Conversely, if coil %M0007 is off because the fault occurred on a different module, its normally closed contact is on and the shutdown occurs.


FST_SCN
MOVE_
INT
CONST
0001
IN
Q
%R0600
LEN
0001
IO_PRES
SVC_
REQ
CONST
0015
FNC
%R0600
PARM
%T0001
%T0001
EQ_
INT
%R0603
I1
Q
CONST
0109
I2
%M0007
%T0001
EQ_
INT
%R0603
I1
Q
CONST
0265
I2
%M0007
IO_PRES
%M0007
SVC_
REQ
CONST
0013
FNC
%R0001
PARM

SVCREQ #16: Read Elapsed Time Clock

Use the SVCREQ function with function number 16 in order 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.

This function has an output parameter block only. The parameter block has a length of 3 words.

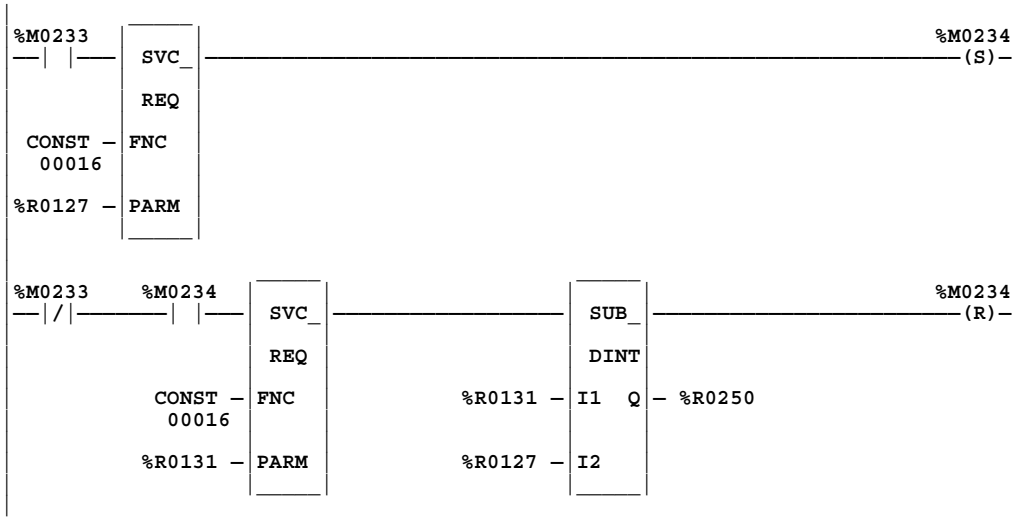
seconds from power on (low order)	address
seconds from power on (high order)	address + 1
100 microsecond ticks	address + 2

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 %M0233 is on, the value of the elapsed time clock is read and coil %M0234 is set. When it is off, the value is read again. The difference between the values is then calculated, and the result is stored in register memory at location %R0250.

The parameter block for the first read is at %R0127; for the second read, at %R0131. 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.



SVCREQ #18: Read I/O Override Status

Use SVCREQ function #18 in order to read the current status of overrides in the CPU.

Note

This feature is available ***only*** for 331 or higher CPUs.

For this function, the parameter block has a length of 1 word. It is an output parameter block only.

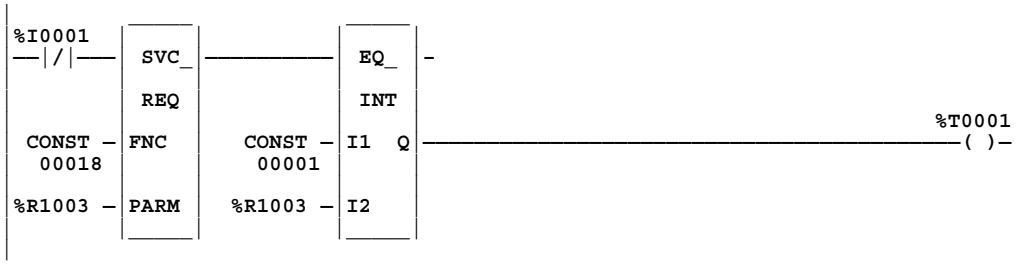
0 = No overrides are set. 1 = Overrides are set.	address

Note

SVCREQ #18 reports only overrides of %I and %Q references.

Example

In the following example, the status of I/O overrides is always read into location %R1003. If any overrides are present, output %T0001 is set on.



SVCREQ #23: Read Master Checksum

Use SVCREQ function #23 to read the master checksums for the user program and the configuration. The SVCREQ output is always set to ON if the function is enabled, and the output block of information (see below) starts at the address given in parameter 3 (PARM) of the SVCREQ function.

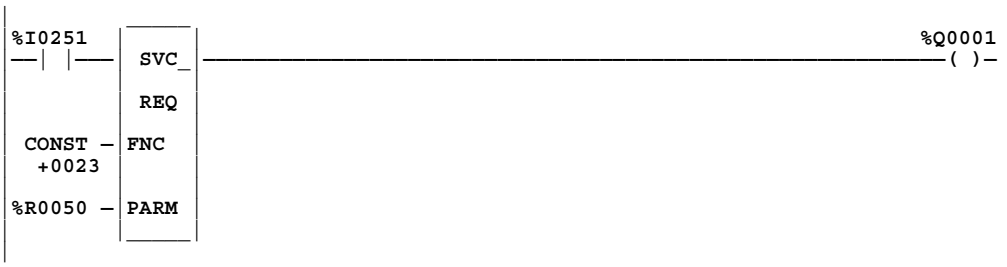
When a **RUN MODE STORE** is active, the program checksums may not be valid until the store is complete. Therefore, two flags are provided at the beginning of the output parameter block to indicate when the program and configuration checksums are valid.

For this function, the output parameter block has a length of 12 words with this format:

Master Program Checksum Valid (0 = not valid, 1 = valid)	address
Master Configuration Checksum Valid (0 = not valid, 1 = valid)	address + 1
Number of Program Blocks (including _MAIN)	address + 2
Size of User Program in Bytes (DWORD data type)	address + 3
Program Additive Checksum	address + 5
Program CRC Checksum (DWORD data type)	address + 6
Size of Configuration Data in Bytes	address + 8
Configuration Additive Checksum	address + 9
Configuration CRC Checksum (DWORD data type)	address + 10

Example

In the following example, when input %I0251 is ON, the master checksum information is placed into the parameter block, and the output coil (%Q0001) is turned on. The parameter block is located at %R0050.



SVCREQ #24: Reset Smart Module

Use SVCREQ function #24 to reset a daughterboard or smart module. The SVCREQ output is set ON unless an invalid number for rack and/or slot is entered as shown below.

For this function, the parameter block has a length of 1 word. It is an input parameter block only.

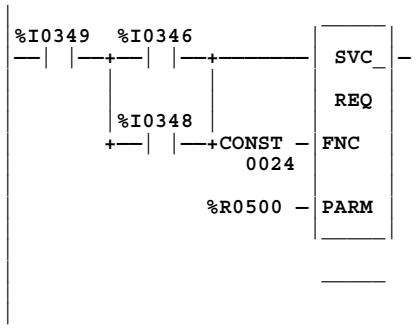
Module Slot (low byte) Module Rack (high byte)	address

Note: Rack 0, Slot 1 shall indicate a reset is to be sent to the daughterboard.

Example

In the following example, when input %I0346 is on and input %I0349 is on, the module indicated by the Rack/Slot present in %R0500 is reset.

The parameter block containing the modules rack and slot for the reset module Service Request is located at %R0500. The parameter block is set up elsewhere in the program.



SVCREQ #26/30: Interrogate I/O

Use SVCREQ function #26 (or #30—they are identical; i.e., you can use either number to accomplish the same thing) to interrogate the actual modules present and compare them with the rack/slot configuration, generating addition, loss, and mismatch alarms, as if a store configuration had been performed. This SVCREQ will generate faults on both the PLC and I/O fault tables, depending on the fault.

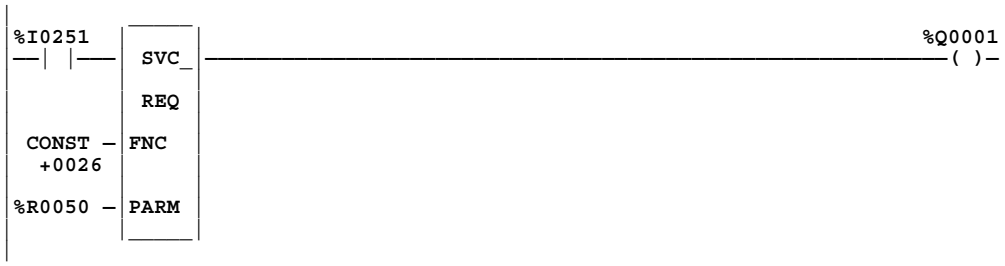
This function has no parameter block and always outputs power flow.

Note

The time for this SVCREQ to execute depends on how many faults exist. Therefore, execution time of this SVCREQ will be greater for situations where more modules are at fault.

Example

In the following example, when input %I0251 is ON, the actual modules are interrogated and compared to the rack/slot configuration. Output %Q0001 is turned on after the SVCREQ is complete.



Note

This Service Request is not available on Micro PLCs.

SVCREQ #29: Read Elapsed Power Down Time

Use the SVCREQ function #29 to read the amount of time elapsed between the last power-down and the most recent power-up. The SVCREQ output is always set to ON, and the output block of information (see below) starts at the address given in parameter 3 (PARM) of the SVCREQ function.

Note

This function is available only in the 331 or higher CPUs.

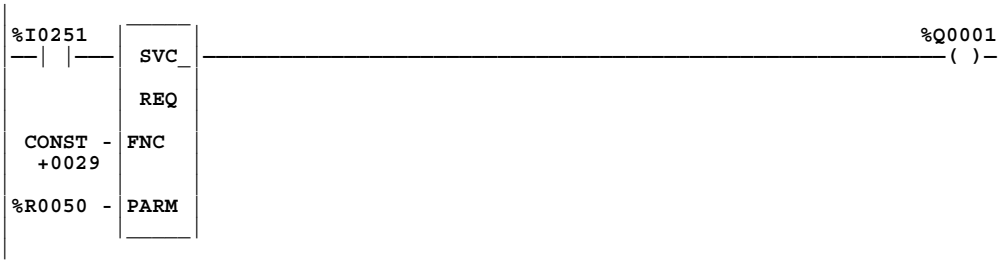
This function has an output parameter block only. The parameter block has a length of 3 words.

Power-Down Elapsed Seconds (low order)	address
Power-Down Elapsed Seconds (high order)	address + 1
100 Microsecond ticks	address + 2

The first two words are the power-down elapsed time in seconds. The last word is the remaining power-down elapsed time in 100 microsecond ticks (which is always 0). Whenever the PLC can not properly calculate the power down elapsed time, the time will be set to 0. This will happen when the PLC is powered up with CLR M/T pressed on the HHP. This will also happen if the watchdog timer times out before power-down.

Example

In the following example, when input %I0251 is ON, the Elapsed Power-Down Time is placed into the parameter block, and the output coil (%Q0001) is turned on. The parameter block is located at %R0050.



SVCREQ #45: Skip Next Output & Input Scan

(Suspend I/O) Use the SVCREQ function #45 to skip the next output and input scans. Any changes to the output reference tables during the sweep in which the SVCREQ #45 was executed will not be reflected on the physical outputs of the corresponding modules. Any changes to the physical input data on the modules will not be reflected in the corresponding input references during the sweep after the one in which the SVCREQ #45 was executed.

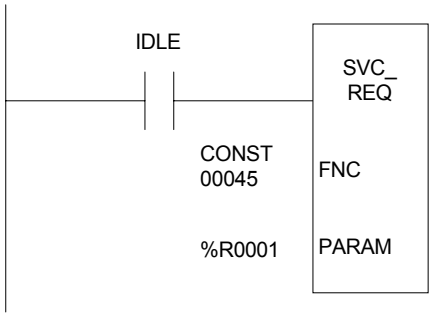
This function has no parameter block.

Note

The DOIO Function Block is not affected by the use of SVCREQ #45. It will still update the I/O when used in the same logic program as the SVCREQ #45.

Example

In the following example, when the "Idle" contact passes power flow, the next Output and Input Scan are skipped.



SVCREQ #46: Fast Backplane Status Access

This function is a method of communicating a few bits to or from one or more smart modules very quickly across the PLC backplane compared with the normal communication method. This increase in communication speed is achieved by limiting the amount of data and the number of replies.

Use SVCREQ function #46 to perform one of the following fast backplane access functions:

- Read a word of extra status data from one of more specified smart modules.
- Write a word of extra status data from one of more specified smart modules.
- Read/Write: Read a word of extra status data from one or more specified modules and write the data value between 0 and 15 to the same module, all in one operation.

Notes

Currently, the only module designed to support this Service Request is the DSM314 (Digital Servo Module).

A COMM_REQ or DOIO function block should not be performed with the specified module(s) during the same logic sweep during which either of the data write functions are performed, since they can cause the write data to be lost.

Two functions that write to a module (Write or Read/Write) should not be performed with the same module during the same logic sweep because they can cause the first write data to be lost.

This Service Request is also known as “SNAP.”

This Service Request has a variable length as described below. The first word of the parameter block determines which function will be used and has the following format:

1 = Read extra data 2 = Write extra data 3 = Read/write extra data	address (word 1)

Read Extra Status Data (Function #1)

The Read Extra Data function reads a word of extra status data from each of the modules specified by a list in the parameter block and places the status data values into the parameter block. The parameter block requires (N + 4) words of reference memory, where N is the number of modules to which the data will be written.

Use the table on the following page to interpret the output values.

Table 12-5. Parameter Block for Read Extra Data Function

Location	Field	Meaning
Address	Function	1 = read extra status data
Address + 1	Error Code	An error code is placed here if the function fails because any of the modules is not present, inappropriate, or not working. For details, see “Error Codes” on page 12-75.
Address + 2	Error rack & slot	The rack & slot number at which the error occurred
Address + 3	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the 1st module from which the data will be read
Address + 4	Read data from first module	The data read from the first module will be place here
Address + 5	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the 2nd module from which the data will be read
Address + 6	Read data from second module	The data read from the second module will be place here
Address + (I * 2) + 1	Ith rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the Ith module from which the data will be read
Address + (I * 2) + 2	Read data from Ith module	The data read from the Ith module will be place here
Address + (N * 2) + 1	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module from which the data will be read
Address + (N * 2) + 2	Read data from last module	The data read from the last module will be place here
Address + (N * 2) + 3	End of list indicator	A zero in this word indicates the end of the list of modules

Write Data (Function #2)

The write data function writes a data value between 0 and 15 from the parameter block to one or more modules specified by a list in the parameter block. The parameter block requires (N + 4) words of reference memory, where N is the number of modules to which the data will be written.

Table 12-6. Parameter Block for Write Data Function

Location	Field	Meaning
Address	Function	2 = write data
Address + 1	Error Code	An error code is placed here if the function fails because any of the modules is not present, inappropriate, or not working. No error code is set if the function executes but any of the modules does not receive the write data properly. For details, see "Error Codes" on page 12-75.
Address + 2	Error rack & slot	The rack & slot number at which the error occurred
Address + 3	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the 1st module to which the data will be sent
Address + 4	Write data for first module	This data value will be written to the first module
Address + 5	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the 2nd module to which the data will be sent
Address + 6	Write data for second module	This data value will be written to the second module
Address + (I * 2) + 1	Ith rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the Ith module to which the data will be sent
Address + (I * 2) + 2	Write data for Ith module	This data value will be written to the Ith module
Address + (N * 2) + 1	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module to which the data will be sent
Address + (N * 2) + 2	Write data for last module	This data value will be written to the last module
Address + (N * 2) + 3	End of list indicator	A zero in this word indicates the end of the list of modules

Read/Write Data (Function #3)

The read/write function reads a word of extra status data from a module specified in the parameter block, then writes a data value between 0 and 15 from the parameter block to that module. This read write process is repeated for each module in a list in the parameter block. The parameter block (N * 3) + 3 words of reference memory, where N is the number of modules with which data will be exchanged.

Table 12-7. Parameter Block for Read/Write Data Function

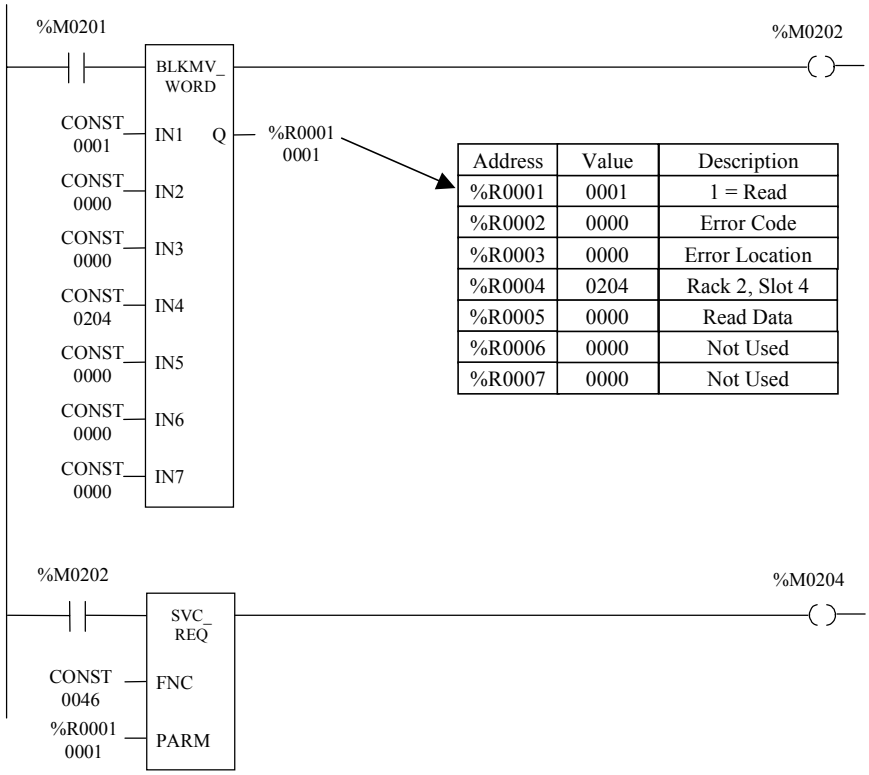
Location	Field	Meaning
Address	Function	3 = read/write
Address + 1	Error Code	An error code is placed here if the function fails because any of the modules is not present, inappropriate, or not working. No error code is set if the function executes but any of the modules does not receive the write data properly. For details, see "Error Codes" on page 12-75.
Address + 2	Error rack & slot	The rack & slot number at which the error occurred
Address + 3	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the 1st module with which data will be exchanged
Address + 4	Read data from first module	The data read from the first module will be placed here
Address + 5	Write data for first module	This data value will be written to the first module
Address + 6	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the 2nd module with which data will be exchanged
Address + 7	Read data from second module	The data read from the second module will be placed here
Address + 8	Write data for second module	This data value will be written to the second module
Address + ((I-1) * 3) + 3	Ith rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the Ith module with which data will be exchanged
Address + ((I-1) * 3) + 4	Read data from Ith module	The data read from the Ith module will be placed here
Address + ((I-1) * 3) + 5	Write data for Ith module	This data value will be written to the Ith module
Address + ((N-1) * 3) + 3	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module with which data will be exchanged
Address + ((N-1) * 3) + 4	Read data from last module	The data read from the last module will be placed here
Address + ((N-1) * 3) + 5	Write data for last module	This data value will be written to the last module
Address + (N * 3) + 3	End of list indicator	A zero in this word indicates the end of the list of modules

Table 12-8. Error Codes

Value	Description
1	Success — the function has executed normally.
-1	Module not present in the specified slot.
-2	Module inappropriate — module in the specified slot is not a smart module or does not support this functionality.
-3	Module not working — module in the specified slot is not communicating with the CPU properly.
-4	Read data parity error — parity error occurred during a read operation from an expansion or remote rack.
-5	Invalid function specified in the command block.

Example 1

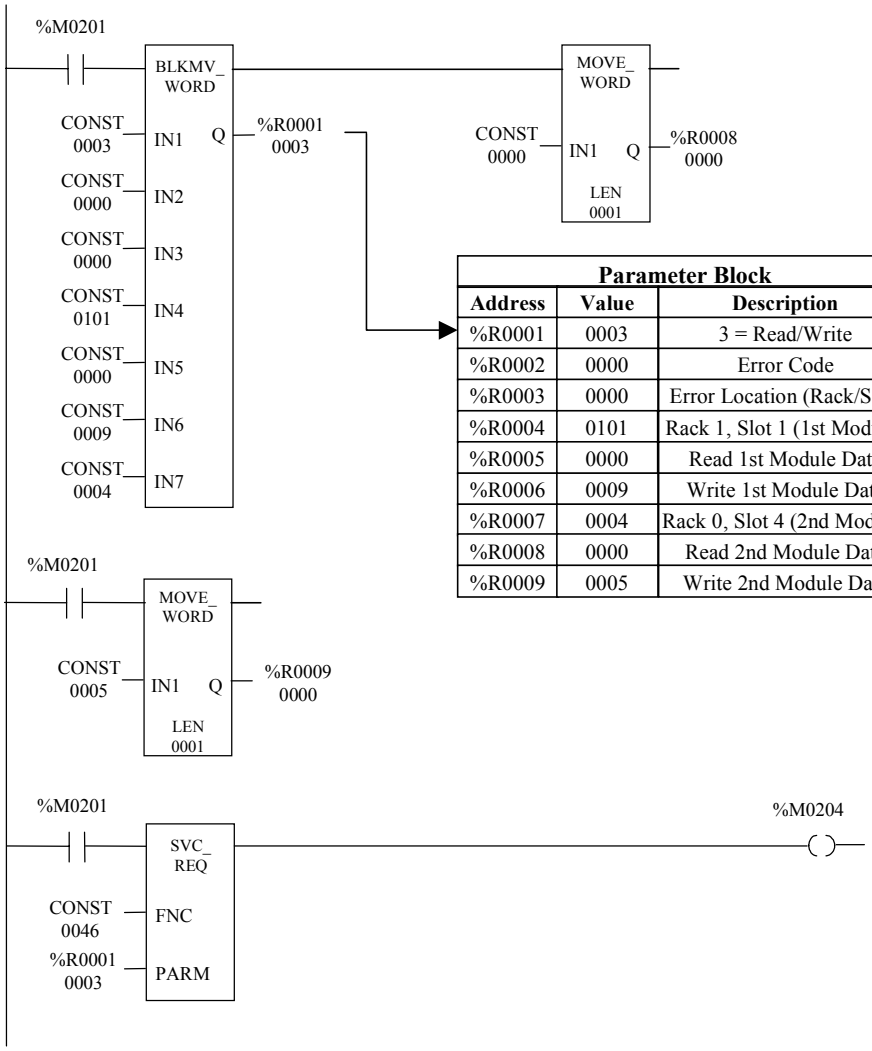
The following example shows a Read (Specified in %R0001) of a single module, located at Rack 2, Slot 4 (specified in %R0004). If the function completes successfully, the data read will be stored in %R0005. If an error occurs, however, an error code will be written to %R0002, and the rack/slot location of the module generating the error will appear in %R0003. Note that since this is a Read function for a single module, Address + 5 and Address + 6 are not used. Therefore, the corresponding memory locations, %R0006 and %R0007, are filled with zeros from the BLKMV instruction's IN6 and IN7 inputs. If an additional module were to be read, %R0006 and %R0007 would be used for the additional module. For more information on the Read function, see Table 12-5 earlier in this chapter.



Address	Value	Description
%R0001	0001	1 = Read
%R0002	0000	Error Code
%R0003	0000	Error Location
%R0004	0204	Rack 2, Slot 4
%R0005	0000	Read Data
%R0006	0000	Not Used
%R0007	0000	Not Used

Example 2

In this example the BLKMV and two MOVE instructions write the required data to the parameter block, which starts at %R0001 (specified by the SVCREQ PARM input). When enabled, the SVCREQ reads the extra status word data from the module in Rack 0, Slot 4 and from the module in Rack 1, Slot 1. It writes a value of 0005 to the module in Rack 0, Slot 4,and a value of 0009 to the module in Rack 1, Slot 1. (Note that the modules do not need to be listed in the parameter block in order by slot numbers.) Data read from the module in Rack 0, Slot 4 will be placed into %R0008. Data read from the module in Rack 1, Slot 1 will be placed in %R0005.



Parameter Block

Address	Value	Description
%R0001	0003	3 = Read/Write
%R0002	0000	Error Code
%R0003	0000	Error Location (Rack/Slot)
%R0004	0101	Rack 1, Slot 1 (1st Module)
%R0005	0000	Read 1st Module Data
%R0006	0009	Write 1st Module Data
%R0007	0004	Rack 0, Slot 4 (2nd Module)
%R0008	0000	Read 2nd Module Data
%R0009	0005	Write 2nd Module Data

SVCREQ #48: Reboot After Fatal Fault Auto Reset

Compatibility for SVCREQ 48

CPU - This Service Request is supported by firmware release 10.00 (or later version) for Series 90-30 CPUs 331, 340, 341, 350, 36x, and 37x.

Software - This Service Request is only supported by VersaPro Version 1.1 (or later version) PLC software. Logicmaster does not support this feature.

Warning

The Reboot After Fatal Fault feature should not be used (Ignore Fatal Faults parameter set to Disabled) in applications where an automatic PLC restart under fault condition could produce an unsafe condition in the controlled equipment. It is the responsibility of the system designer to determine whether this feature can be used safely with their equipment. Failure to follow this warning could result in injury or death to personnel and/or damage to equipment.

Description

The Reboot After Fatal Fault Service Request lets the PLC automatically resume normal operation after a fatal fault has occurred. Following the fatal fault, the PLC will automatically reset and resume execution. The faults will not be cleared, but will be treated as non-fatal. If fatal faults are present following the power up, the PLC will still be allowed to transition to run mode. This feature is enabled by the Ignore Fatal Faults (or Fatal Fault Override) parameter in the CPU's hardware configuration.

SVCREQ 48 sets the maximum number of retries and the time period during which the retries may occur. If the number of retries allowed within the time period is exceeded, the CPU mode is set to STOP/FAULT. If the period is 0, the CPU mode is set to STOP/FAULT when the number of retries allowed is exceeded.

If the operator cycles power, fatal faults are ignored. The current fault count and time period are initialized. The total number of fatal faults is unchanged, but the total number of retries is incremented. System bit %S0021 is set to 1 whenever retry is successful and remains set until all fatal faults are cleared, or the mode is set to STOP/FAULT.

Table 12-9. Parameter Block for Reboot after Fatal Fault

Location	Field	Meaning
Word 1	Service Request Status	See Return Status Definition, below. User program must initialize this word to zero.
Word 2	Unlimited Retries	0 = Disable (number of retries is set by Word 3) 1 = Enable (Words 3 and 4 ignored)
Word 3	Number of Retries Allowed	Range is 0 to 128 0 = Automatic Reboot is Disabled 1 to 128 = Maximum number of retries that are allowed to occur within the period set in Word 4.
Word 4	Retry Period (in minutes)	Range is 0 to 5940 minutes (99 hours) 0 = No time limit on maximum number of retries set in Word 3. Auto Reboot will be allowed for the number of retries. 1 to 5940 = Auto Reboot is disabled if the number of retries specified is exceeded within the period specified.

Table 12-10. Return Status Definitions for Reboot after Fatal Fault

Status	Description	Notes	Power Flow
-5	Invalid Retry Period	Valid range is 0 to 5940	No
-4	Invalid No. of Retries	Valid range is 0 to 128	No
-3	Invalid Unlimited Retries	Must be 0 or 1	No
-2	Configuration Disabled	Ignore Fatal Faults (Fatal Fault Override) option must be enabled in hardware configuration.	No
0	No Action	Command requires no change	Yes
1	Auto Reset Enabled	Valid command enables reboot after Fatal Fault	Yes
2	Auto Reset Disabled	Valid command disables Reboot after Fatal Fault. Ignore Fatal Faults remains enabled.	

SVCREQ 49 Auto Reset Statistics

Service Request 49 provides access to two variables which record total number of fatal faults and retires that have occurred. The range of these variables is 0 to 65535. These variables do not roll over if their maximum value is exceeded. (Service Request 48 is used to configure the maximum number of retries allowed and the time limit during which the retries can occur.)

Table 12-11. Parameter Block for Auto Reset Statistics

Word 1	Service Request Status	See Return Status Definitions below. User program must initialize this word to zero.
Word 2	Command	0 = Return total number of Fatal Faults and Number of Retries that have occurred. 1 = Initialize the Total Number of Fatal Faults and Total Number of Retries to Zero.
Word 3	Returned Value = Total number of Fatal Faults that have occurred.	User program should initialize to zero.
Word 4	Returned Value = Total number of Auto Reset Retries	User program should initialize to zero.

Table 12-12. Return Status Definitions for Auto Reset Statistics

Status	Description	Notes	Power Flow
-2	Configuration Disabled	Ignore Fatal Faults (Fatal Fault Override) option must be enabled in hardware configuration.	No
-1	Invalid Command	Command must be 0 or 1.	No
1	Normal Status	Valid Command	Yes

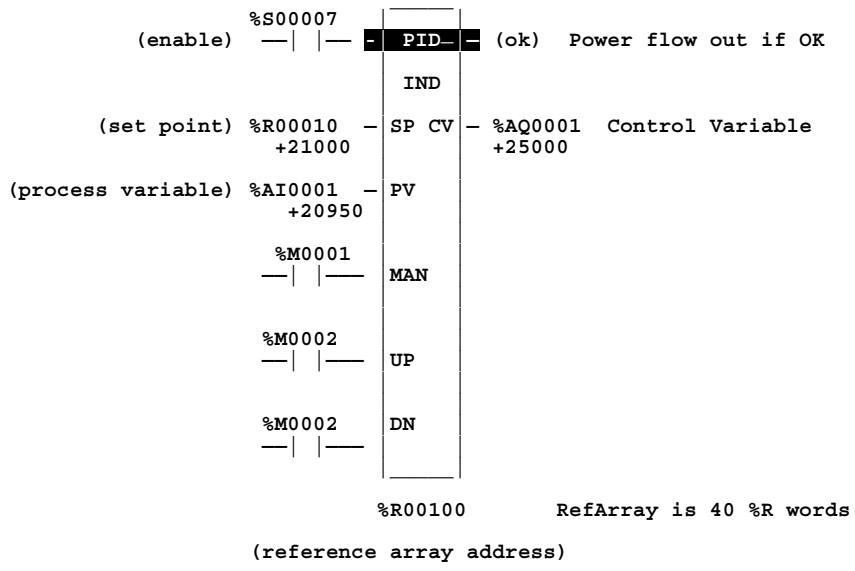
CPU Compatibility for SVCREQ 49

This Service Request is supported by Firmware Release 10.00 for the Series 90-30 CPUs 331, 340, 341, 350, 36x, and 37x.

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 (PV) feedback with a desired process Set Point (SP) and updates a Control Variable (CV) 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 12-82) 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.



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 because the PID block calculates the error from the difference of these two inputs. The PV and CV counts can 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 less than 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 this parameter is not energized (0), 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.

*Increments (UP parameter) or decremented (DN parameter) by 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 can 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 12-13. 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 12-80).
%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 Parameter Details" table on page 12-85)
%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 12-13. 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	Non-configurable
%Ref+0024		maintained by	
%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

The RefArray array must consist of %R registers on the 90-30 PLC. 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 operating 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 12-87) 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 12-3. The number in parentheses after each parameter name is the offset in the RefArray.

Table 12-14. PID Parameter 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 90-30/20/Micro 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. 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 90-30 will not use a solution time less than 10 milliseconds (see Note on page 12-80); 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 might 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 12-14. PID Parameter 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 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 12-87) 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 6.50 or later 90-30 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 12-14. PID Parameter 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 use 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 12-3 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 might 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 12-3 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.

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 (Reverse Acting), which can be changed to Direct Acting mode (PV – SP) by setting the Error Term to 1. The Error Term is the low bit (0-bit) in the Config. Word (%Ref+0012). In a Direct Acting proportional (P) loop, an increase in the Process Variable (PV) causes an increase in the output (CV). In a Reverse Acting proportional (P) loop, an increase in the Process Variable (PV) causes a decrease in the output (CV). Introducing the integral term (I) changes the behavior. In a Direct Acting PI loop, the output (CV) will increase when the process variable (PV) is greater than the setpoint (SP). In a Reverse Acting PI loop, the output (CV) will decrease when the Process Variable (PV) is greater than the Setpoint (SP).

Direct Acting: Error = measurement – setpoint (PV-SP), Error Term = 1

Reverse Acting: Error = setpoint – measurement (SP-PV), Error Term = 0

Note. **Direct Acting** is sometimes referred to as **Forward Acting**.

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:

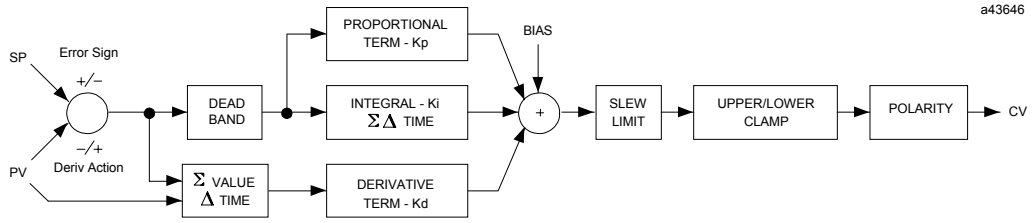


Figure 12-4. Independent Term Algorithm (PIDIND)

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

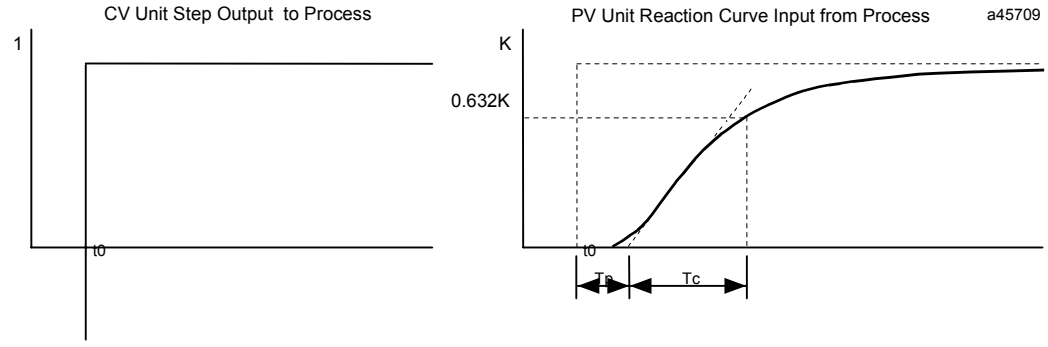
The PID loop gains, Kp, Ki and Kd, are determined by the characteristics of the process being controlled. Two key questions when setting up a PID loop are:

1. How big is the change in PV when CV changes 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 functional block 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

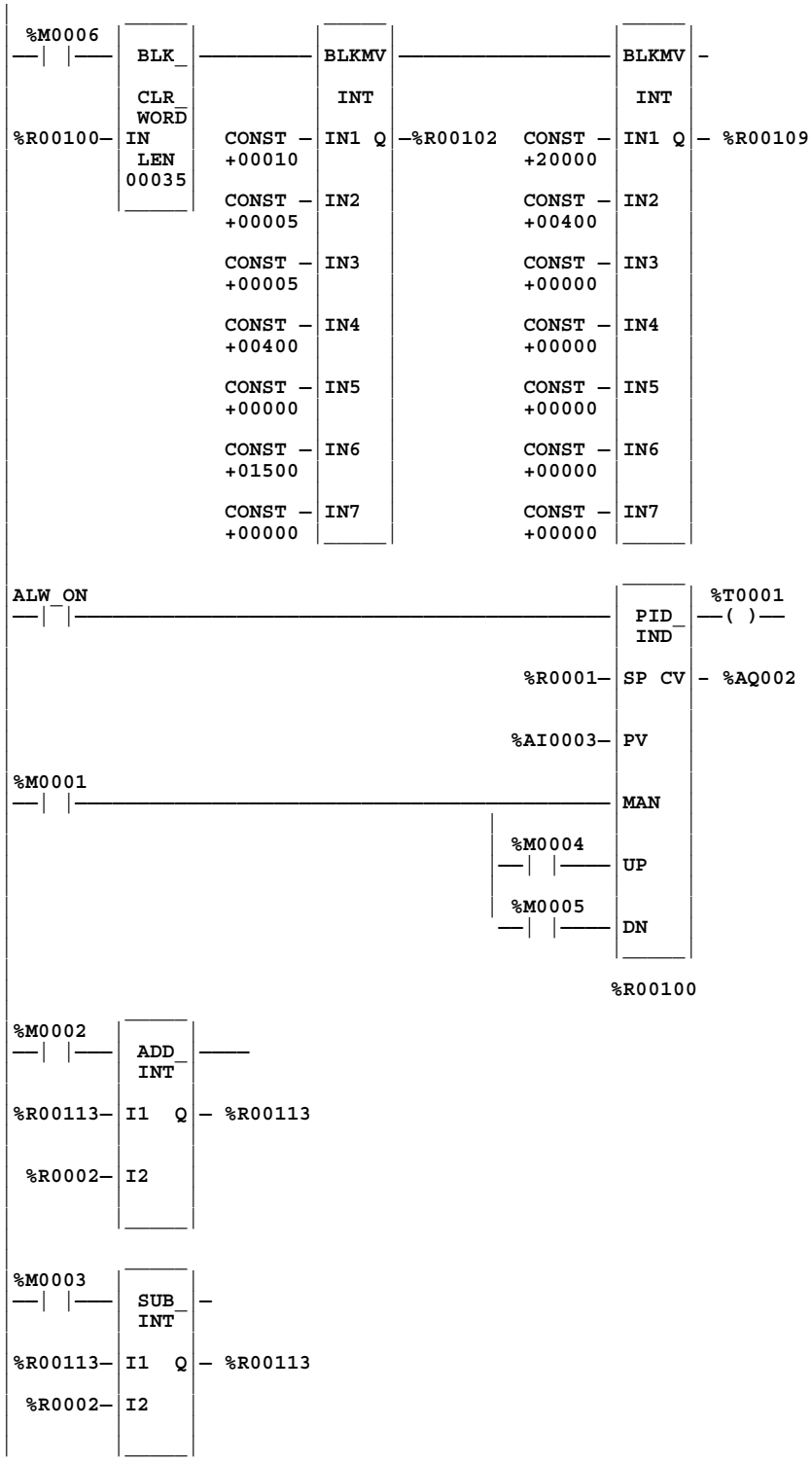
Once initial gains are determined, they must be converted to integer User Parameters. To avoid scaling problems, the Process gain, K, should be calculated as a change in input PV Counts divided by the output step change in CV Counts and not in process PV or CV engineering units. All times should also 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 milliseconds, 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. Closing the %M0006 contact enables a pair of BLKMV instructions, which set the initial parameter values by copying constants into the 14 words starting at %R102 (%Ref+2). (Note: to optimize parameters during the tuning process, access parameters by placing the Logicmaster cursor on the PID instruction and pressing the F10 key, which is the Zoom key.)

The block can be switched to Manual mode with %M0001 so that the Manual Command, %R0113, can be adjusted. Bits %M0004 or %M0005 can be used to increase or decrease %R0113 and the PID CV and integrator by 1 every 100 millisecond solution. For faster manual operation, bits %M0002 and %M0003 can be used to add or subtract the value in %R0002 to/from %R0113 every PLC sweep. The %T0001 output is on when the PID is OK. Note that some of the registers in the 40-register parameter block are not included either because they are not used in this example, or they are not configurable because they are used by the PLC system. For additional parameter information, see Table 12-8.

Address	Value	Description
%R0102	+00010	Sample Period
%R0103	+00005	Dead Band +
%R0104	+00005	Dead Band –
%R0105	+00400	Proportional Gain (Kp)
%R0106	+00000	Derivative Gain (Kd)
%R0107	+01500	Integral Gain (Ki)
%R0108	+00000	CV Bias/Output Offset
%R0109	+20000	Upper Clamp
%R0110	+00400	Lower Clamp
%R0111	+00000	Minimum Slew Time
%R0112	+00000	Config. Word
%R0113	+00000	Manual Command
%R0114	+00000	Control Word
%R0115	+00000	Internal SP (Non-Configurable)


%M0006
BLK_
CLR_
WORD
%R00100
IN
LEN
00035
CONST +00010
CONST +00005
CONST +00005
CONST +00400
CONST +00000
CONST +01500
CONST +00000
BLKMV
INT
IN1 Q
IN2
IN3
IN4
IN5
IN6
IN7
%R00102
CONST +20000
CONST +00400
CONST +00000
CONST +00000
CONST +00000
CONST +00000
BLKMV
INT
IN1 Q
IN2
IN3
IN4
IN5
IN6
IN7
%R00109
ALW_ON
PID_
IND
%T0001
%R0001
SP CV
%AQ002
%AI0003
PV
%M0001
MAN
%M0004
UP
%M0005
DN
%R00100
%M0002
ADD_
INT
%R00113
I1 Q
%R00113
%R0002
I2
%M0003
SUB_
INT
%R00113
I1 Q
%R00113
%R0002
I2

Appendix A Instruction Timing

The Series 90-30, 90-20, and Micro PLCs support many different functions and function blocks. This appendix contains tables showing the memory size in bytes and the execution time in microseconds for each function. Memory size is the number of bytes required by the function in a ladder diagram application program.

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) and not in the discrete memory.
Disabled	Time required to execute the function when power flows into the function or function block; however, it is in an inactive state, as when a timer is held in the reset state.

Note

Timers and counters are updated each time they are encountered in the logic, timers by the amount of time consumed by the last sweep and counters by one count.

Note

For the 350, 351, 352, and 360 PLC CPUs, times are identical except for the MOVE instruction, which is different for the 350 CPU—refer to the note at the bottom of the table on page A-6.

Table A-1. Instruction Timing, Standard Models

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
Timers	On-Delay Timer	146	81	80	42	105	39	38	21	–	–	–	–	15
	Off-Delay Timer	98	47	44	23	116	63	58	32	–	–	–	–	9
	Timer	122	76	75	40	103	54	53	30	–	–	–	–	15
Counters	Up Counter	137	70	69	36	130	63	62	33	–	–	–	–	11
	Down Counter	136	70	69	37	127	61	61	31	–	–	–	–	11
Math	Addition (INT)	76	47	46	24	41	0	1	0	–	–	–	–	13
	Addition (DINT)	90	60	60	34	41	1	0	0	–	–	–	–	13
	Subtraction (INT)	75	46	45	25	41	0	1	0	–	–	–	–	13
	Subtraction (DINT)	92	62	62	34	41	1	0	0	–	–	–	–	13
	Multiplication (INT)	79	49	50	28	41	0	1	0	–	–	–	–	13
	Multiplication (DINT)	108	80	101	43	41	1	0	0	–	–	–	–	13
	Division (INT)	79	51	50	27	41	0	1	0	–	–	–	–	13
	Division (DINT)	375	346	348	175	41	1	0	0	–	–	–	–	13
	Modulo Division (INT)	78	51	49	27	41	0	1	0	–	–	–	–	13
	Modulo Div (DINT)	134	103	107	54	41	1	0	0	–	–	–	–	13
	Square Root (INT)	153	124	123	65	42	0	1	0	–	–	–	–	9
	Square Root (DINT)	268	239	241	120	42	0	0	1	–	–	–	–	9
Relational	Equal (INT)	66	35	36	19	41	1	1	0	–	–	–	–	9
	Equal (DINT)	86	56	54	29	41	1	0	0	–	–	–	–	9
	Not Equal (INT)	67	39	35	22	41	1	1	0	–	–	–	–	9
	Not Equal (DINT)	81	51	51	28	41	1	0	0	–	–	–	–	9
	Greater Than (INT)	64	33	35	20	41	1	1	0	–	–	–	–	9
	Greater Than (DINT)	89	59	58	32	41	1	0	0	–	–	–	–	9
	Greater Than/Eq (INT)	64	36	34	19	41	1	1	0	–	–	–	–	9
	Greater Than/Eq (DINT)	87	58	57	30	41	1	0	0	–	–	–	–	9
	Less Than (INT)	66	35		19	41	1	1	0	–	–	–	–	9
	Less Than (DINT)	87	57		30	41	1	1	0	–	–	–	–	9
	Less Than/Equal (INT)	66	36	34	21	41	1	1	0	–	–	–	–	9
	Less Than/Equal (DINT)	86	57	56	31	41	1	1	0	–	–	–	–	9
	Range (INT)	92	58	54	29	46	1	0	1	–	–	–	–	15
	Range(DINT)	106	75	57	37	45	0	0	0	–	–	–	–	15
	Range(WORD)	93	60	54	29	0	0	0	0	–	–	–	–	15

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.

Table A-1. Instruction Timing, Standard Models-Continued

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
Bit Operation	Logical AND	67	37	37	22	42	0	0	1	–	–	–	–	13
	Logical OR	68	38	38	21	42	0	0	1	–	–	–	–	13
	Logical Exclusive OR	66	38	37	20	42	0	1	1	–	–	–	–	13
	Logical Invert, NOT	62	32	31	17	42	0	1	1	–	–	–	–	9
	Shift Bit Left	139	89	90	47	74	26	23	13	11.61	11.61	12.04	6.29	15
	Shift Bit Right	135	87	85	45	75	26	24	13	11.63	11.62	12.02	6.33	15
	Rotate Bit Left	156	127	126	65	42	1	1	0	11.70	11.78	12.17	6.33	15
	Rotate Bit Right	146	116	116	62	42	1	1	0	11.74	11.74	12.13	6.27	15
	Bit Position	102	72	49	38	42	1	0	0	–	–	–	–	13
	Bit Clear	68	38	35	21	42	1	1	1	–	–	–	–	13
	Bit Test	79	49	51	28	41	0	0	1	–	–	–	–	13
	Bit Set	67	37	37	20	42	0	0	0	–	–	–	–	13
	Masked Compare (WORD)	217	154	141	74	107	44	39	21	–	–	–	–	25
	Masked Compare (DWORD)	232	169	156	83	108	44	39	22	–	–	–	–	25
Data Move	Move (INT)	68	37	39	20	43	0	0	0	1.62	1.62	5.25	1.31	13
	Move (BIT)	94	62	64	35	42	0	0	0	12.61	12.64	12.59	6.33	13
	Move (WORD)	67	37	40	20	41	0	0	0	1.62	1.63	5.25	1.31	13
	Block Move (INT)	76	48	50	28	59	30	30	16	–	–	–	–	27
	Block Move (WORD)	76	48	49	29	59	29	28	15	–	–	–	–	27
	Block Clear	56	28	27	14	43	0	0	0	1.35	1.29	1.40	0.78	9
	Shift Register (BIT)	201	153	153	79	85	36	34	18	0.69	0.68	0.71	0.37	15
	Shift Register (WORD)	103	53	52	29	73	25	23	12	1.62	1.62	2.03	1.31	15
	Bit Sequencer	165	101	99	53	96	31	29	16	0.07	0.07	0.08	0.05	15
	COMM_REQ	1317	1272	1489	884	41	2	0	0	–	–	–	–	13
Table	Array Move													
	INT	230	201	177	104	72	41	40	20	1.29	1.15	10.56	2.06	21
	DINT	231	202	181	105	74	44	42	23	3.24	3.24	10.53	2.61	21
	BIT	290	261	229	135	74	43	42	23	–.03	–.03	-0.01	0.79	21
	BYTE	228	198	176	104	74	42	42	23	0.81	0.82	8.51	1.25	21
	WORD	230	201	177	104	72	41	40	20	1.29	1.15	10.56	2.06	21
	Search Equal													
	INT	197	158	123	82	78	39	37	20	1.93	1.97	2.55	1.55	19
	DINT	206	166	135	87	79	38	36	21	4.33	4.34	4.55	2.44	19
	BYTE	179	141	117	74	78	38	36	21	1.53	1.49	1.83	1.03	19
	WORD	197	158	123	82	78	39	37	20	1.93	1.97	2.55	1.55	19

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-1. Instruction Timing, Standard Models-Continued

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
	Search Not Equal													
	INT	198	159	124	83	79	39	36	21	1.93	1.93	2.48	1.52	19
	DINT	201	163	132	84	79	37	35	21	6.49	6.47	6.88	3.82	19
	BYTE	179	141	117	73	79	38	36	19	1.54	1.51	1.85	1.05	19
	WORD	198	159	124	83	79	39	36	21	1.93	1.93	2.48	1.52	19
	Search Greater Than													
	INT	198	160	125	82	79	37	38	19	3.83	3.83	4.41	2.59	19
	DINT	206	167	135	88	78	38	36	20	8.61	8.61	9.03	4.88	19
	BYTE	181	143	118	73	79	37	36	19	3.44	3.44	3.75	2.03	19
	WORD	198	160	125	82	79	37	38	19	3.83	3.83	4.41	2.59	19
	Search Greater Than/Eq													
	INT	197	160	124	83	77	38	36	20	3.86	3.83	4.45	2.52	19
	DINT	205	167	136	87	80	39	36	21	8.62	8.61	9.02	4.87	19
	BYTE	180	142	118	75	79	37	37	20	3.47	3.44	3.73	2.00	19
	WORD	197	160	124	83	77	38	36	20	3.86	3.83	4.45	2.52	19
	Search Less Than													
	INT	199	159	124	84	78	38	36	20	3.83	3.86	4.48	2.48	19
	DINT	206	168	135	87	79	38	38	19	8.62	8.60	-1.36	4.88	19
	BYTE	181	143	119	75	80	38	37	20	3.44	3.44	3.75	2.00	19
	WORD	199	159	124	84	78	38	36	20	3.83	3.86	4.45	2.48	19
	Search Less Than/Equal													
	INT	200	158	124	82	79	38	37	21	3.79	3.90	4.45	2.55	19
	DINT	207	167	137	88	78	39	37	19	8.60	8.61	9.01	4.86	19
	BYTE	180	143	119	74	78	40	37	19	3.46	3.44	3.73	2.02	19
	WORD	200	158	124	82	79	38	37	21	3.79	3.90	4.45	2.55	19
Conversion	Convert to INT	74	46	39	25	42	1	1	1	–	–	–	–	9
	Convert to BCD–4	77	50	34	25	42	1	1	1	–	–	–	–	9

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-1. Instruction Timing, Standard Models-Continued

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
Control	Call a Subroutine	155	93	192	85	41	0	0	0	–	–	–	–	7
	Do I/O	309	278	323	177	38	1	0	0	–	–	–	–	12
	PID – ISA Algorithm	1870	1827	1812	929	91	56	82	30	–	–	–	–	15
	PID – IND Algorithm	2047	2007	2002	1017	91	56	82	30	–	–	–	–	15
	End Instruction	–	–	–	–	–	–	–	–	–	–	–	–	–
	Service Request													
	# 6	93	54	63	45	41	2	0	0	–	–	–	–	9
	# 7 (Read)	–	37	309	161	–	2	0	0	–	–	–	–	9
	# 7 (Set)	–	37	309	161	–	2	0	0	–	–	–	–	9
	#14	447	418	483	244	41	2	0	0	–	–	–	–	9
	#15	281	243	165	139	41	2	0	0	–	–	–	–	9
	#16	131	104	115	69	41	2	0	0	–	–	–	–	9
	#18	–	56	300	180	–	2	0	0	–	–	–	–	9
	#23	1689	1663	1591	939	43	1	0	0	–	–	–	–	9
	#26//30*	1268	1354	6680	3538	42	0	0	0	–	–	–	–	9
	#29	–	–	55	41	–	–	1	0	–	–	–	–	9
	Nested MCR/ENDMCR Combined	135	73	68	39	75	25	21	12	–	–	–	–	8

*Service request #26/30 was measured using a high speed counter, 16-point output, in a 5-slot rack.

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-2. Instruction Timing, 35x-36x Models

Function Group	Function	Enabled 350/351/36x	Disabled 350/351/36x	Increment 350/351/36x	Enabled 352	Disabled 352	Increment 352	Size
Timers	On-Delay Timer	4	6	–	4	5	–	15
	Timer	3	3	–	2	2	–	15
	Off-Delay Timer	3	3	–	3	2	–	15
Counters	Up Counter	1	3	–	2	2	–	13
	Down Counter	3	3	–	1	2	–	13
Math	Addition (INT)	2	0	–	1	0	–	13
	Addition (DINT)	2	0	–	2	0	–	19
	Addition (REAL)	52	0	–	33	0	–	17
	Subtraction (INT)	2	0	–	1	0	–	13
	Subtraction (DINT)	2	0	–	2	0	–	19
	Subtraction (REAL)	53	0	–	34	0	–	17
	Multiplication (INT)	21	0	–	21	0	–	13
	Multiplication (DINT)	24	0	–	24	0	–	19
	Multiplication (REAL)	68	1	–	38	1	–	17
	Division (INT)	22	0	–	22	0	–	13
	Division (DINT),	25	0	–	25	0	–	19
	Division (REAL)	82	2	–	36	2	–	17
	Modulo Division (INT)	21	0	–	21	0	–	13
	Modulo Div (DINT)	25	0	–	25	0	–	19
	Square Root (INT)	42	1	–	41	1	–	10
	Square Root (DINT)	70	0	–	70	0	–	13
	Square Root (REAL)	137	0	–	35	0	–	11
Trigonometric	SIN (REAL)	360	0	–	32	0	–	11
	COS (REAL)	319	0	–	29	0	–	11
	TAN (REAL)	510	1	–	32	1	–	11
	ASIN (REAL)	440	0	–	45	0	–	11
	ACOS (REAL)	683	0	–	63	0	–	11
	ATAN (REAL)	264	1	–	33	1	–	11
Logarithmic	LOG (REAL)	469	0	–	32	0	–	11
	LN (REAL)	437	0	–	32	0	–	11
Exponential	EXP	639	0	–	42	0	–	11
	EXPT	89	1	–	54	1	–	17
Radian Conversion	Convert RAD to DEG	65	1	–	32	1	–	11
	Convert DEG to RAD	59	0		32	0		11

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for Model 351 and 352 CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.

Table A-2. Instruction Timing, 35x-36x Models-Continued

Function Group	Function	Enabled 350/351/36x	Disabled 350/351/36x	Increment 350/351/36x	Enabled 352	Disabled 352	Increment 352	Size
Relational	Equal (INT)	1	0	–	1	0	–	10
	Equal (DINT)	2	0	–	2	0	–	16
	Equal (REAL)	57	0	–	28	0	–	14
	Not Equal (INT)	1	0	–	1	0	–	10
	Not Equal (DINT)	1	0	–	1	0	–	16
	Not Equal (REAL)	62	0	–	31	0	–	14
	Greater Than (INT)	1	0	–	1	0	–	10
	Greater Than (DINT)	1	0	–	1	0	–	16
	Greater Than (REAL)	57	0	–	32	0	–	14
	Greater Than/Equal (INT)	1	0	–	1	0	–	10
	Greater Than/Equal (DINT)	1	0	–	1	0	–	10
	Greater Than/Equal (REAL)	57	1	–	31	1	–	14
	Less Than (INT)	1	0	–	1	0	–	10
	Less Than (DINT)	1	0	–	1	0	–	16
	Less Than (REAL)	58	1	–	36	1	–	14
	Less Than/Equal (INT)	1	0	–	1	0	–	10
	Less Than/Equal (DINT)	3	0	–	3	0	–	16
	Less Than/Equal (REAL)	37	0	–	37	0	–	14
	Range (INT)	2	1	–	2	1	–	13
	Range (DINT)	2	1	–	2	1	–	22
	Range (WORD)	1	0	–	1	0	–	13
Bit Operation	Logical AND	2	0	–	2	0	–	13
	Logical OR	2	0	–	2	0	–	13
	Logical Exclusive OR	1	0	–	1	0	–	13
	Logical Invert, NOT	1	0	–	1	0	–	10
	Shift Bit Left	31	1	1.37	31	1	1.37	16
	Shift Bit Right	28	0	3.03	28	0	3.03	16
	Rotate Bit Left	25	0	3.12	25	0	3.12	16
	Rotate Bit Right	25	0	4.14	25	0	4.14	16
	Bit Position	20	1	–	20	1	–	13
	Bit Clear	20	0	–	20	0	–	13
	Bit Test	20	0	–	20	0	–	13
	Bit Set	19	1	–	19	1	–	13
	Mask Compare (WORD)	52	0	–	52	0	–	25
	Mask Compare (DWORD)	50	0	–	49	0	–	25

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for Model 351 and 352 CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-2. Instruction Timing, 35x-36x Models-Continued

Function Group	Function	Enabled 350/351/36X	Disabled 350/351/36X	Increment 350/351/36X	Enabled 352	Disabled 352	Increment 352	Size
Data Move	Move (INT)	2	0	0.41	2	0	0.41	10
	Move (BIT)	28	0	4.98	28	0	4.98	13
	Move (WORD)	2	0	0.41	2	0	0.41	10
	Move (REAL)	24	1	0.82	24	1	0.82	13
	Block Move (INT)	2	0	–	2	0	–	28
	Block Move (WORD)	4	4	–	3	0	–	28
	Block Move (REAL)	41	0	–	41	0	–	13
	Block Clear	1	0	0.24	1	0	0.24	11
	Shift Register (BIT)	49	0	0.23	46	0	0.23	16
	Shift Register (WORD)	27	0	0.41	27	0	0.41	16
	Bit Sequencer	38	22	0.02	38	22	0.02	16
	COMM_REQ	765	0	–	765	0	–	13
Table	Array Move							
	INT	54	0	0.97	54	0	0.97	22
	DINT	54	0	0.81	54	0	0.81	22
	BIT	69	0	0.36	69	0	0.36	22
	BYTE	54	1	0.64	54	1	0.64	22
	WORD	54	0	0.97	54	0	0.97	22
	Search Equal							
	INT	37	0	0.62	37	0	0.62	19
	DINT	41	1	1.38	41	1	1.38	22
	BYTE	35	0	0.46	35	0	0.46	19
	WORD	37	0	0.62	37	0	0.62	19
	Search Not Equal							
	INT	37	0	0.62	37	0	0.62	19
	DINT	38	0	2.14	38	0	2.14	22
	BYTE	37	0	0.47	37	0	0.47	19
	WORD	37	0	0.62	37	0	0.62	19
	Search Greater Than							
	INT	37	0	1.52	37	0	1.52	19
	DINT	39	0	2.26	39	0	2.26	22
	BYTE	36	1	1.24	36	1	1.24	19
	WORD	37	0	1.52	37	0	1.52	19
	Search Greater Than/Equal							
	INT	37	0	1.48	37	0	1.48	19
	DINT	39	0	2.33	39	0	2.33	22
	BYTE	37	1	1.34	37	1	1.34	19
	WORD	37	0	1.48	37	0	1.48	19

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for 350 and 360 Series CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-2. Instruction Timing, 35x-36x Models-Continued

Function Group	Function	Enabled 350/351/36x	Disabled 350/351/36x	Increment 350/351/36x	Enabled 352	Disabled 352	Increment 352	Size
	Search Less Than							
	INT	37	0	1.52	37	0	1.52	19
	DINT	41	1	2.27	41	1	2.27	22
	BYTE	37	0	1.41	37	0	1.41	19
	WORD	37	0	1.52	37	0	1.52	19
	Search Less Than/Equal							
	INT	38	0	1.48	38	0	1.48	19
	DINT	40	1	2.30	40	1	2.30	22
	BYTE	37	0	1.24	37	0	1.24	19
	WORD	38	0	1.48	38	0	1.48	19
Conversion	Convert to INT	19	1	–	19	1	–	10
	Convert to BCD-4	21	1	–	21	1	–	10
	Convert to REAL	27	0	–	21	0	–	8
	Convert to WORD	28	1	–	30	1	–	11
	Truncate to INT	32	0	–	32	0	–	11
	Truncate to DINT	63	0	–	31	0	–	11
Control	Call a Subroutine	72	1	–	73	1	–	7
	Do I/O	114	1	–	115	1	–	13
	PID – ISA Algorithm*	162	34	–	162	34	–	16
	PID – IND Algorithm*	146	34	–	146	34	–	16
	End Instruction	–	–	–	–	–	–	–
	Service Request							
	#6	22	1	–	22	1	–	10
	#7 (Read)	75	1	–	75	1	–	10
	#7 (Set)	75	1	–	75	1	–	10
	#14	121	1	–	121	1	–	10
	#15	46	1	–	46	1	–	10
	#16	36	1	–	36	1	–	10
	#18	261	1	–	261	1	–	10
	#23	426	0	–	426	0	–	10
	#26//30**	2260	1	–	2260	1	–	10
	#29	20	0	–	20	0	–	10
	#43							
	Nested MCR/ENDMCR Combined	1	1	–	1	1	–	4
	Sequential Event Recorder (SER)	See Table A-3	26.50	See Table A-3				

*The PID times shown above are based on the 6.5 release of the 351 CPU.

**Service request #26/30 was measured using a high speed counter, 16-point output, in a 5-slot rack.

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for 350 and 360 Series CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-3. SER Function Block Timing

Configuration	Example	Time (µsec)
No power flow (disabled)	—	26.50
Contiguous		
8 channels	%I1—8	79.94
16 channels	%I1—16	80.58
24 channels	%I1—24	81.56
32 channels	%I1—32	81.73
8 + 8 contiguous channels	%I1—8 and %Q1—8	111.03
8 + 8 + 8 contiguous channels	%I1—8, %Q1—8 and %M1—8	143.38
8 + 8 + 8 + 8 contiguous channels	%I1—8, %Q1—8 and %M1—8 and %T1—8	175.79
Noncontiguous	%I1, %M10, %Q3, etc.	
8 channels		299.64
16 channels		552.83
24 channels		806.35
32 channels		1059.85
Reset		
with 8 channels	—	162.63
with 16 channels	—	267.51
with 24 channels	—	372.73
with 32 channels	—	477.95

Notes: When a slot with an Input module is specified add an additional 46 µsecs to each of the **Contiguous** and **Noncontiguous** timings.

When the trigger occurs, add an additional 29 usec if using BCD format or 148 usec if using Posix format.

Times shown for reset are for the maximum buffer size of 1024 samples. (Reset clears all samples in the sample buffer.)

Table A-4. Instruction Timing, 37x Models

Function Group	Function	Enabled 37x	Disabled 37x	Increment 37x	Size
Timers	On-Delay Timer	4	5	–	15
	Timer	2	2	–	15
	Off-Delay Timer	3	2	–	15
Counters	Up Counter	2	2	–	13
	Down Counter	1	2	–	13
Math	Addition (INT)	1	0	–	13
	Addition (DINT)	2	0	–	19
	Addition (REAL)	5	0	–	17
	Subtraction (INT)	1	0	–	13
	Subtraction (DINT)	2	0	–	19
	Subtraction (REAL)	5	0	–	17
	Multiplication (INT)	5	0	–	13
	Multiplication (DINT)	5	0	–	19
	Multiplication (REAL)	5	0	–	17
	Division (INT)	5	0	–	13
	Division (DINT),	5	0	–	19
	Division (REAL)	5	0	–	17
	Modulo Division (INT)	5	0	–	13
	Modulo Div (DINT)	5	0	–	19
	Square Root (INT)	5	0	–	10
	Square Root (DINT)	10	0	–	13
	Square Root (REAL)	5	0	–	11
Trigonometric	SIN (REAL)	10	0	–	11
	COS (REAL)	10	0	–	11
	TAN (REAL)	10	0	–	11
	ASIN (REAL)	10	0	–	11
	ACOS (REAL)	10	0	–	11
	ATAN (REAL)	5	0	–	11
Logarithmic	LOG (REAL)	5	0	–	11
	LN (REAL)	5	0	–	11
Exponential	EXP	10	0	–	11
	EXPT	10	0	–	17
Radian Conversion	Convert RAD to DEG	5	0	–	11
	Convert DEG to RAD	5	0		11

Notes:
1. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
2. Enabled time for single length units of type %R, %AI, and %AQ.
3. COMMREQ time has been measured between CPU and HSC.
4. DOIO is the time to output values to discrete output module.
5. Where there is more than one possible case, the time indicated above represents the worst possible case.

Table A-4. Instruction Timing, 37x Models- Continued

Function Group	Function	Enabled 37x	Disabled 37x	Increment 37x	Size
Relational	Equal (INT)	1	0	–	10
	Equal (DINT)	2	0	–	16
	Equal (REAL)	5	0	–	14
	Not Equal (INT)	1	0	–	10
	Not Equal (DINT)	1	0	–	16
	Not Equal (REAL)	5	0	–	14
	Greater Than (INT)	1	0	–	10
	Greater Than (DINT)	1	0	–	16
	Greater Than (REAL)	5	0	–	14
	Greater Than/Equal (INT)	1	0	–	10
	Greater Than/Equal (DINT)	1	0	–	10
	Greater Than/Equal (REAL)	5	0	–	14
	Less Than (INT)	1	0	–	10
	Less Than (DINT)	1	0	–	16
	Less Than (REAL)	5	0	–	14
	Less Than/Equal (INT)	1	0	–	10
	Less Than/Equal (DINT)	3	0	–	16
	Less Than/Equal (REAL)	5	0	–	14
	Range (INT)	2	0	–	13
	Range (DINT)	2	0	–	22
	Range (WORD)	1	0	–	13
Bit Operation	Logical AND	2	0	–	13
	Logical OR	2	0	–	13
	Logical Exclusive OR	1	0	–	13
	Logical Invert, NOT	1	0	–	10
	Shift Bit Left	5	0	1	16
	Shift Bit Right	5	0	1	16
	Rotate Bit Left	5	0	1	16
	Rotate Bit Right	5	0	1	16
	Bit Position	5	0	–	13
	Bit Clear	5	0	–	13
	Bit Test	5	0	–	13
	Bit Set	5	0	–	13
	Mask Compare (WORD)	9	0	–	25
	Mask Compare (DWORD)	10	0	–	25

Notes:
1. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
2. Enabled time for single length units of type %R, %AI, and %AQ.
3. COMMREQ time has been measured between CPU and HSC.
4. DOIO is the time to output values to discrete output module.
5. Where there is more than one possible case, the time indicated above represents the worst possible case.

Table A-4. Instruction Timing, 37x Models- Continued

Function Group	Function	Enabled 37x	Disabled 37x	Increment 37x	Size
Data Move	Move (INT)	2	0	1	10
	Move (BIT)	5	0	1	13
	Move (WORD)	2	0	1	10
	Move (REAL)	5	0	1	13
	Block Move (INT)	2	0	–	28
	Block Move (WORD)	3	0	–	28
	Block Move (REAL)	11	1	–	13
	Block Clear	1	0	1	11
	Shift Register (BIT)	10	0	1	16
	Shift Register (WORD)	15	0	1	16
	Bit Sequencer	14	10	1	16
	COMM_REQ	200	200	–	13
Table	Array Move				
	INT	10	0	1	22
	DINT	15	0	1	22
	BIT	10	0	1	22
	BYTE	10	0	1	22
	WORD	10	0	1	22
	Search Equal				
	INT	5	0	1	19
	DINT	5	0	2	22
	BYTE	5	0	1	19
	WORD	5	0	1	19
	Search Not Equal				
	INT	5	0	1	19
	DINT	10	0	2	22
	BYTE	5	0	2	19
	WORD	5	0	2	19
	Search Greater Than				
	INT	5	0	1	19
	DINT	5	0	2	22
	BYTE	10	0	1	19
	WORD	5	0	1	19
	Search Greater Than/Equal				
	INT	5	0	1	19
	DINT	5	0	2	22
	BYTE	5	0	1	19
	WORD	5	0	1	19

Notes:
1. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
2. Enabled time for single length units of type %R, %AI, and %AQ.
3. COMMREQ time has been measured between CPU and HSC.
4. DOIO is the time to output values to discrete output module.
5. Where there is more than one possible case, the time indicated above represents the worst possible case.
6. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-4. Instruction Timing, 37x Models- Continued

Function Group	Function	Enabled 37x	Disabled 37x	Increment 37x	Size
	Search Less Than				
	INT	5	0	1	19
	DINT	10	0	2	22
	BYTE	5	0	1	19
	WORD	5	0	1	19
	Search Less Than/Equal				
	INT	5	0	1	19
	DINT	5	0	2	22
	BYTE	5	0	1	19
	WORD	5	0	1	19
Conversion	Convert to INT	5	0	–	10
	Convert to BCD-4	5	0	–	10
	Convert to REAL	5	0	–	8
	Convert to WORD	5	0	–	11
	Truncate to INT	5	0	–	11
	Truncate to DINT	5	0	–	11
Control	Call a Subroutine	15	0	–	7
	Do I/O	5	0	–	13
	PID – ISA Algorithm	14	10	–	16
	PID – IND Algorithm	14	10	–	16
	End Instruction	–	–	–	–
	Service Request				
	#6	5	0	–	10
	#7 (Read)	10	0	–	10
	#7 (Set)	5	0	–	10
	#14	15	0	–	10
	#15	5	0	–	10
	#16	10	0	–	10
	#18	255	0	–	10
	#23	25	0	–	10
	#26//30**	155	0	–	10
	#29	5	0	–	10
	Nested MCR/ENDMCR Combined	1	0	–	4
	Sequential Event Recorder (SER) 8 Channels	60	0	=	
	Sequential Event Recorder (SER) 16 Channels	199	0	=	

**Service request #26/30 was measured using a high speed counter, 16-point output, in a 5-slot rack.

Notes:
1. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
2. Enabled time for single length units of type %R, %AI, and %AQ.
3. COMMREQ time has been measured between CPU and HSC.
4. DOIO is the time to output values to discrete output module.
5. Where there is more than one possible case, the time indicated above represents the worst possible case.
6. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

CPU Boolean Execution Times

This table lists execution times of coils and contacts for the Series 90-30 CPU modules.

Table A-5. Boolean Execution Times

CPU Model	Execution Time per 1,000 Boolean Contacts/Coils
Models 37x	0.15 milliseconds
Models 35x and 36x	0.22 milliseconds
Models 340/341	0.3 milliseconds
Model 331	0.4 milliseconds
Models 313/323	0.6 milliseconds
Model 311	18.0 milliseconds

Instruction Sizes for CPUs 350 - 374

Memory Size in the following table refers to the number of bytes of user memory required by a given instruction in a ladder diagram application program.

Table A-6. Instruction Sizes for CPUs 350 – 374

Function	Memory Size
Pop stack and AND to top	1
Pop stack and OR to top	1
Duplicate top of stack	1
Pop stack	1
Initial stack	1
Label	5
Jump	5
All other instructions	3
Function blocks – see Table A-2	various

Appendix B Interpreting Fault Tables

The Series 90-30, Series 90-20, and Series 90 Micro PLCs maintain 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 fault data in these tables only exists in the PLC, not in the folder. Therefore, if using Logicmaster, you must be connected in either the ONLINE or MONITOR mode to view faults.

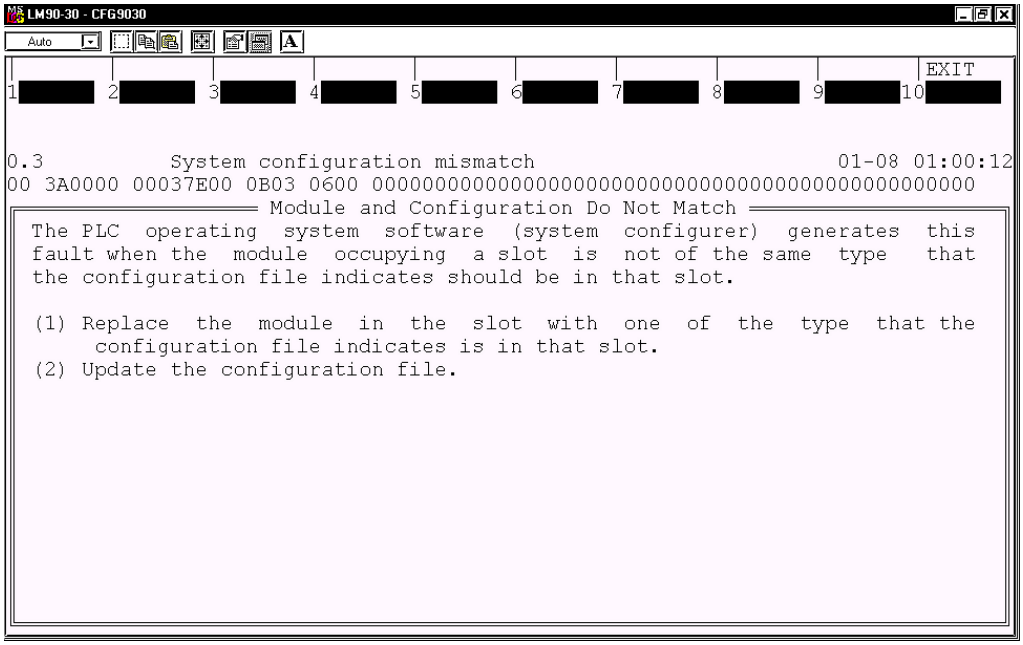
- The PLC fault table contains:
  - Fault location.
  - Fault description.
  - Date and time of fault.
- The I/O fault table contains:
  - Fault location.
  - Reference address.
  - Fault category.
  - Fault type.
  - Date and time of fault.

PLC Fault Table

Access the PLC fault table through the programming software. For information about accessing fault tables, refer to the online help, *Logicmaster 90 Series 90-30/20/Micro Programming Software User's Manual*, GFK-0466.

Example

The following figure shows the example of a System Configuration Mismatch fault that has been zoomed to its fault detail screen.



The following diagram identifies each field in the fault entry for the System Configuration Mismatch fault displayed above:

This System Configuration Mismatch fault entry is explained below. (All data is in hexadecimal.)

Field	Value	Comment
Long/Short	00	00=Short, which indicates only 8 bytes of the Fault Extra Data field are used
Rack	00	Main rack (rack 0)
Slot	03	Slot 3
Task	F2	
Fault Group	0B	System Configuration Mismatch fault
Fault Action	03	FATAL fault
Error Code	06	
Fault Extra Data	00	No Fault Extra Data reported in the 8 bytes

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 8 bytes or all 24 bytes of the Fault Extra Data field are used.

Type	Code	Fault Extra Data
Short	00	8 bytes
Long	01	24 bytes

Spare

These six 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 an expansion cable.

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 gives additional information for PLC engineers; typically, the task 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 Logicmaster 90-30/20/Micro software is based on the fault group and the error codes.

Table B-1 lists the possible fault groups in the PLC fault table.

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	Hexadecimal		
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
132	84	Password access failure	Informational
135	87	PLC CPU software failure	Fatal
137	89	PLC sequence-store failure	Fatal

Fault Action

Each fault may have one of three actions associated with it. These fault actions are fixed on the Series 90-30 PLC and cannot be changed by the user.

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. Table B-3 shows error codes for the PLC Software Error Group (Group 87H).

Table B-3. Alarm Error Codes for PLC CPU Software Faults

Decimal	Hexadecimal	Name
20	14	Corrupted PLC Program Memory
39	27	Corrupted PLC Program Memory
82	52	Backplane Communications Failed
90	5A	User Shut Down Requested
All others		PLC CPU Internal System Error

Table B-4 shows the error codes for all the other fault groups.

Table B-4. Alarm Error Codes for PLC Faults

Decimal	Hexadecimal	Name
PLC Error Codes for Loss of Option Module Group (4)		
44	2C	Option Module Soft Reset Failed
45	2D	Option Module Soft Reset Failed
255	FF	Option Module Communication Failed
79	4F	Loss of Daughterboard
Error Codes for Reset of, Addition of, or Extra Option Module Group (8)		
2	2	Module Restart Complete
04	4	Addition of Daughterboard
05	5	Reset of Daughterboard
	All others	Reset of, Addition of, or Extra Option Module
Error Codes for Option Module Software Failure Group (10 hex)		
1	1	Unsupported Board Type
2	2	COMREQ – mailbox full on outgoing message that starts the COMREQ
3	3	COMREQ – mailbox full on response
5	5	Backplane Communications with PLC; Lost Request
11	B	Resource (alloc, tbl ovrflw, etc.) error
13	D	User program error
401	191	Module Software Corrupted; Requesting Reload
Error Codes for System Configuration Mismatch Group (B hex)		
8	8	Analog Expansion Mismatch
10	A	Unsupported Feature
23	17	Program exceeds memory limits
58	3A	Mismatch of Daughterboard
Error Codes for System Bus Error Group (C hex)		
All others		System Bus Error
Error Codes for Program Block Checksum Group (11 hex)		
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
Error Codes for User Application Fault Group (16 hex)		
2	2	PLC Watchdog Timer Timed Out
5	5	COMREQ – WAIT mode not available for this command
6	6	COMREQ – Bad Task ID
7	7	Application Stack Overflow
Error Codes for System Bus Failure Group (80 hex)		
1	1	Operating system
Error Codes for Corrupted User RAM on Powerup Group (82 hex)		
1	1	Corrupted User RAM on Power-up
2	2	Illegal Boolean Opcode Detected
3	3	PLC_ISCP_PC_OVERFLOW
4	4	PRG_SYNTAX_ERR
Error Codes for PLC CPU Hardware Faults (D hex)		
All codes		PLC CPU Hardware Failure

Fault Extra Data

This field contains details of the fault entry. The following example shows what data may be present:

Example - Corrupted User RAM Group

Four of the error codes in the System Configuration Mismatch group supply fault extra data:

Table B-5. PLC Fault Data – Illegal Boolean Opcode Detected

Fault Extra Data	Model Number Mismatch
[0]	ISCP Fault Register Contents
[1]	Bad OPCODE
[2,3]	ISCP Program Counter
[4,5]	Function Number

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.

PLC Fault Time Stamp

PLC CPU Hardware Failure (RAM Failure)

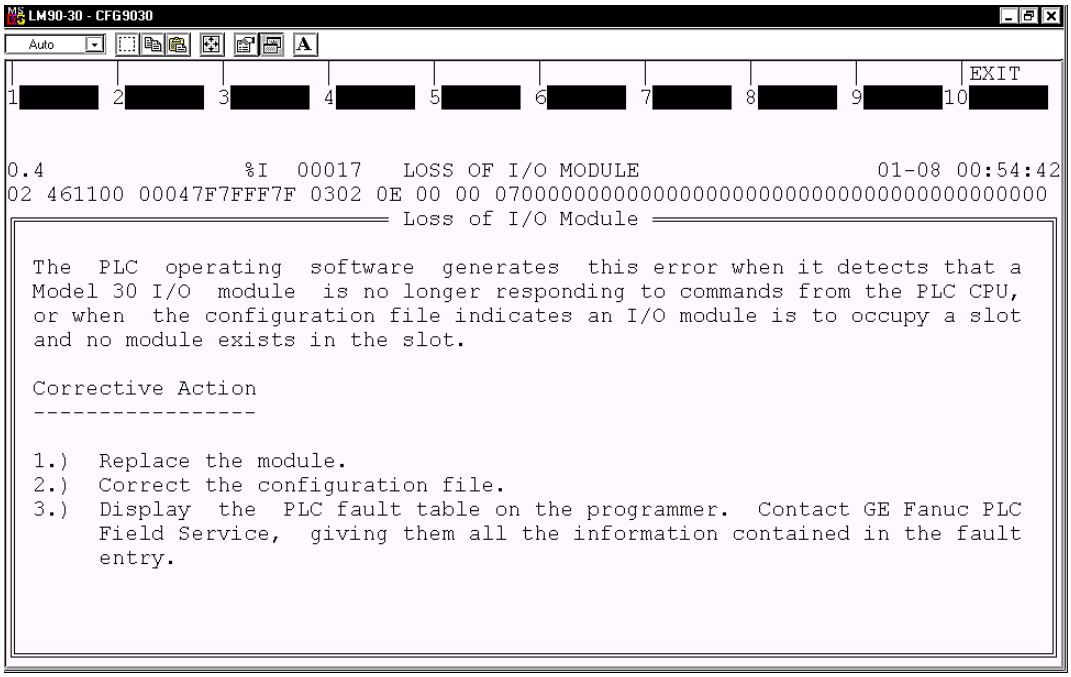
The six-byte time stamp is the value of the system clock when the fault was recorded by the PLC CPU. (Values are coded in BCD format.)

Table B-6. PLC Fault Time Stamp

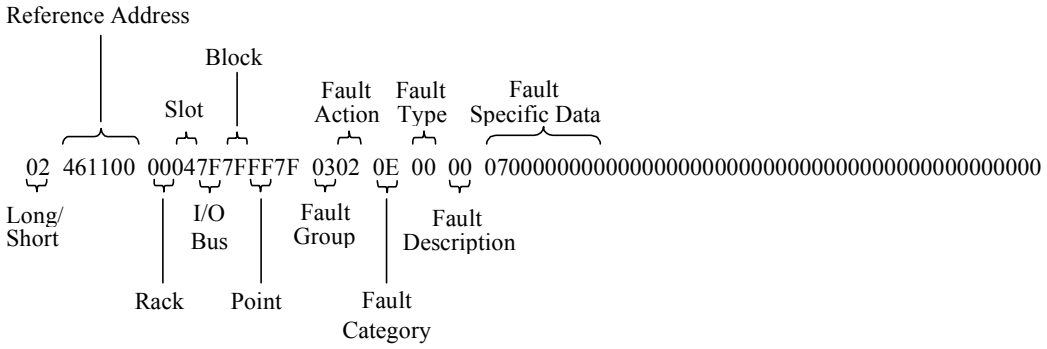
Byte Number	Description
1	Seconds
2	Minutes
3	Hours
4	Day of the month
5	Month
6	Year

I/O Fault Table

The following figure shows the example of a Loss of I/O Module fault that has been zoomed to its fault detail screen.



The following diagram identifies the hexadecimal information displayed in each field in the example fault entry (System Configuration Mismatch) shown in the figure above.



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 particular fault uses 5 bytes or ass 21 bytes of the Fault Specific Data field.

Table B-7. 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-8. I/O Reference Address

Byte	Description	Range
0	Memory Type	0 – FF
1–2	Offset	0 – 7FF

The memory type byte is one of the following values.

Table B-9. 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 that 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, 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, i.e., the one containing the CPU. Racks 1 through 7 are expansion racks.

Slot

The slot number ranges from 0 to 9. The PLC CPU always occupies slot 1 in the main rack (rack 0).

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 Logicmaster 90-30/20/Micro software is based on the fault group and the error codes.

Table B-10 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-10. I/O Fault Groups

Group Number	Group Name	Fault Action
3	Loss of, or missing, I/O module	Diagnostic
7	Addition of, or extra, I/O module	Diagnostic
9	IOC or I/O bus fault	Diagnostic
A	I/O module fault	Diagnostic
–	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. Table B-11 lists possible fault actions.

Table B-11. I/O 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 Specific Data

An I/O fault table entry may contain up to 5 bytes of I/O fault specific data.

Symbolic Fault Specific Data

Table B-12 lists data that is required for block circuit configuration.

Table B-12. I/O Fault Specific Data

Decimal Number	Hex Code	Description
Circuit Configuration		
	1 2 3	Circuit is an input – tristate Circuit is an input Circuit is an output

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-13. I/O Fault Time Stamp

Byte Number	Description
1	Seconds
2	Minutes
3	Hours
4	Day of the month
5	Month
6	Year

Appendix C Instruction Mnemonics

In Program Display/Edit mode, you can quickly enter or search for a programming 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 90-30/20/Micro programming software. The complete mnemonic is shown in column 3 of this table, and the shortest entry you can make for each instruction is listed in column 4.

At any time during programming in Logicmaster, you can display a help screen that lists these mnemonics by pressing the ALT and I keys.

Function Group	Instruction	Mnemonic						
		All	INT	DINT	BIT	BYTE	WORD	REAL
Contacts	Any Contact	&CON	&CON					
	Normally Open Contact	&NOCON	&NOCON					
	Normally Closed Contact	&NCCON	&NCCON					
	Continuation Contact	&CONC	&CONC					
Coils	Any Coil	&COI	&COI					
	Normally Open Coil	&NOCOI	&NOCOI					
	Negated Coil	&NCCOI	&NCCOI					
	Positive Transition Coil	&PCOI	&PCOI					
	Negative Transition Coil	&NCOI	&NCOI					
	SET Coil	&SL	&SL					
	RESET Coil	&RL	&RL					
	Retentive SET Coil	&SM	&SM					
	Retentive RESET Coil	&RM	&RM					
	Retentive Coil	&NOM	&NOM					
	Negated Retentive Coil	&NCM	&NCM					
	Continuation Coil	&COILC	&COILC					
Links	Horizontal Link	&HO	&HO					
	Vertical Link	&VE	&VE					
Timers	On Delay Timer	&ON	&ON					
	Elapsed Timer	&TM	&TM					
	Off Delay Timer	&OF	&OF					
Counters	Up Counter	&UP	&UP					
	Down Counter	&DN	&DN					

Function Group	Instruction	Mnemonic							
		All	**BCD-4**	**INT**	**DINT**	**BIT**	**BYTE**	**WORD**	**REAL**
Math	Addition	&AD		&AD_I	&AD_DI				&AD_R
	Subtraction	&SUB		&SUB_I	&SUB_DI				&SUB_R
	Multiplication	&MUL		&MUL_I	&MUL_DI				&MUL_R
	Division	&DIV		&DIV_I	&DIV_DI				&DIV_R
	Modulo	&MOD		&MOD_I	&MOD_DI				&MOD_R&SQ_R
	Square Root	&SQ		&SQ_I	&SQ_DI				
	Sine	&SIN							
	Cosine	&COS							
	Tangent	&TAN							
	Inverse Sine	&ASIN							
	Inverse Cosine	&ACOS							
	Inverse Tangent	&ATAN							
	Base 10 Logarithm	&LOG							
	Natural Logarithm	&LN							
	Power of e	&EXP							
	Power of x	&EXPT							
Relational	Equal	&EQ		&EQ_I	&EQ_DI				&EQ_R
	Not Equal	&NE		&NE_I	&NE_DI				&NE_R
	Greater Than	&GT		&GT_I	&GT_DI				&GT_R
	Greater or Equal	&GE		&GE_I	&GE_DI				&GE_R
	Less Than	&LT		&LT_I	&LT_DI				&LT_R
	Less Than or Equal	&LE		&LE_I	&LE_DI				&LE_R
Bit Operation	AND	&AN						&AN_W	
	OR	&OR						&OR_W	
	Exclusive OR	&XO						&XO_W	
	NOT	&NOT						&NOT_W	
	Bit Shift Left	&SHL						&SHL_W	
	Bit Shift Right	&SHR						&SHR_W	
	Bit Rotate Left	&ROL						&ROL_W	
	Bit Rotate Right	&ROR						&ROR_W	
	Bit Test	&BT						&BT_W	
	Bit Set	&BS						&BS_W	
	Bit Clear	&BCL						&BCL_W	
	Bit Position	&BP						&BP_W	
	Masked Compare	&MCMP						&MCM_W	
Conversion	Convert to Integer	&TO_INT	&TO_INT_BCD4						
	Convert to Double Integer	&TO_DINT							&BCD4_R
	Convert to BCD–4	&BCD4			&TO_REAL_DI				
	Convert to REAL	&TO_REAL						&TO_REAL_W	
	Convert to WORD	&TO_W							
	Truncate to Integer	&TRINT							
	Truncate to Double Integer	&TRDINT							

Function Group	Instruction	Mnemonic						
		All	INT	DINT	BIT	BYTE	WORD	REAL
Data Move	Move	&MOV	&MOV_I		&MOV_BI		&MOV_W	&MOV_R
	Block Move	&BLKM	&BLKM_I				&BLKM_W	&BLKM_R
	Block Clear	&BLKC						
	Shift Register	&SHF			&SHF_BI			
	Bit Sequencer	&BI					&AR_W	
	Communications Request	&COMMR						
Table	Array Move	&AR	&AR_I	&AR_DI	&AR_BI	&AR_BY	&AR_W	
	Search Equal	&SRCHE	&SRCHE_I	&SRCHE_DI		&SRCHE_BY	&SRCHE_W	
	Search Not Equal	&SRCHN	&SRCHN_I	&SRCHN_DI		&SRCHN_BY	&SRCHN_W	
	Search Greater Than	&SRCHGT	&SRCHGT_I	&SRCHGT_DI		&SRCHGT_BY	&SRCHGT_W	
	Search Greater Than or Equal	&SRCHGE	&SRCHGE_I	&SRCHGE_DI		&SRCHGE_BY	&SRCHGE_W	
	Search Less Than	&SRCHLT	&SRCHLT_I	&SRCHLT_DI		&SRCHLT_BY	&SRCHLT_W	
	Search Less Than or Equal	&SRCHLE	&SRCHLE_I	&SRCHLE_DI		&SRCHLE_BY	&SRCHLE_W_	
Control	Call a Subroutine	&CA						
	Do I/O	&DO						
	SER	&SER						
	PID – ISA Algorithm	&PIDIS						
	PID – IND Algorithm	&PIDIN						
	SFC Reset	&SFCR						
	End	&END						
	Rung Explanation (Comment)	&COMME						
	System Services Request	&SV						
	Master Control Relay	&MCR						
	End Master Control Relay	&ENDMCR						
	Nested Master Control Relay	&MCRN						
	Nested End Master Cntl Relay	&ENDMCRN						
	Jump	&JUMP						
	Nested Jump	&JUMPN						
	Label	&LABEL						
	Nested Label	&LABELN						

Appendix D Key Functions

This appendix lists the keyboard functions that are active in the software environment. To display this information on the Logicmaster screen, press 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-J	Toggle command line.	CTRL-→	Cursor right within the field.
ALT-L	List directory files.	CTRL-D	Decrement reference address.
ALT-P	Print screen.	CTRL-U	Increment reference address.
ALT-H	Help.	Tab	Change/increment field contents.
ALT-K	Key help.	Shift-Tab	Change/decrement field contents.
ALT-I	Instruction mnemonic help.	Enter	Accept field contents.
ALT-N	Toggle display options.	CTRL-E	Display last system error.
ALT-T	Start Teach mode.	F12 or Keypad -	Toggle discrete reference.
ALT-Q	Stop Teach mode.	F11 or Keypad *	Override discrete reference.
ALT-n	Playback file n (n = 0 thru 9).		
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.	CTRL-PgDn	Next rung.
ALT-U	Update disk.	~	Horizontal shunt.
ALT-V	Variable table window.	|	Vertical shunt.
ALT-F2	Go to operand reference table.	Tab	Go to the next operand field.
Special Keys			
ALT-O	Password override. Available only on the Password screen in the configuration software.		

The Help card on the next page contains a listing of the key help and also the instruction mnemonics help text for Logicmaster 90-30/20/Micro software. This card is printed in triplicate and is perforated for easier removal from the manual.

Print side 1 of GFJ-055D on this page.

Print side 2 of GFJ-055D on this page.

Appendix E Using Floating-Point Numbers

There are a few considerations you need to understand when using floating-point numbers. The first section discusses these general considerations. Refer to page E-5 and following for instructions on entering and displaying floating-point numbers.

Note

Floating-point capabilities are ***only*** supported on the 35x and 36x series CPUs Release 9.00 or later, and on all releases of CPU352 and 37x series.

Floating-Point Numbers

The programming software provides the ability to edit, display, store, and retrieve numbers with real values. Some functions operate on floating-point numbers. However, to use floating-point numbers with the programming software, you must have a 35x, 36x or 37x series CPU (see Note above). Floating-point numbers are represented in decimal scientific notation, with a display of six significant digits.

Note

In this manual, the terms “floating-point” and “real” are used interchangeably to describe the floating-point number display/entry feature of the programming software.

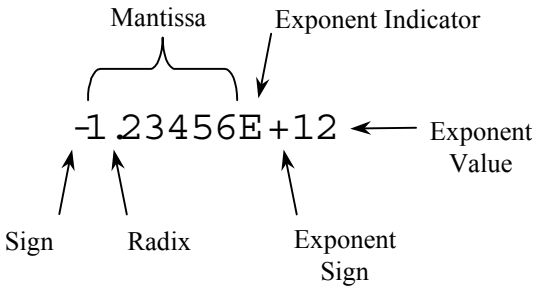
The following format is used. For numbers in the range 9999999 to .0001, the display has no exponent and up to six or seven significant digits. For example:

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 following form: +1.23456E+12

Real Number Terminology

A real number is stored in a 32-bit double word register. The following discusses the terms used for the parts of a real number.



Sign – Either plus or minus. Stored in the most significant bit (bit 32) of the double word. A one in bit 32 indicates a negative sign. A zero in bit 32 indicates a positive sign.

Radix – A period (dot) symbol that separates the whole number portion from the fractional number portion of the mantissa. For decimal numbers, the radix is commonly called the decimal point.

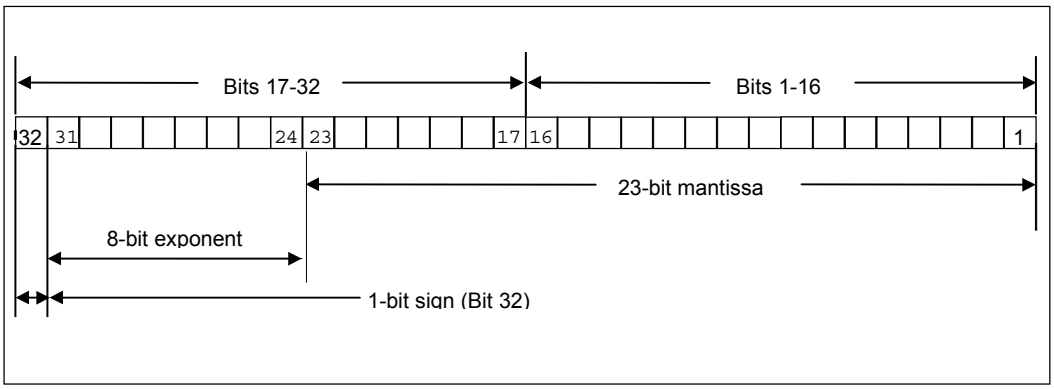
Exponent – (Also called a "Characteristic"). It is stored in 8 bits, in bit positions 31 through 24 of the 32-bit double word. The exponent may have values in the range of +127 to –126; however, the exponent is always stored as a positive number because the CPU automatically adds 127 to its value before storing it.

Mantissa – (Also called a Significand"). The basic number without the sign and exponent. It is stored in 23 bits, in bit positions 23 through 1 of the 32-bit word.

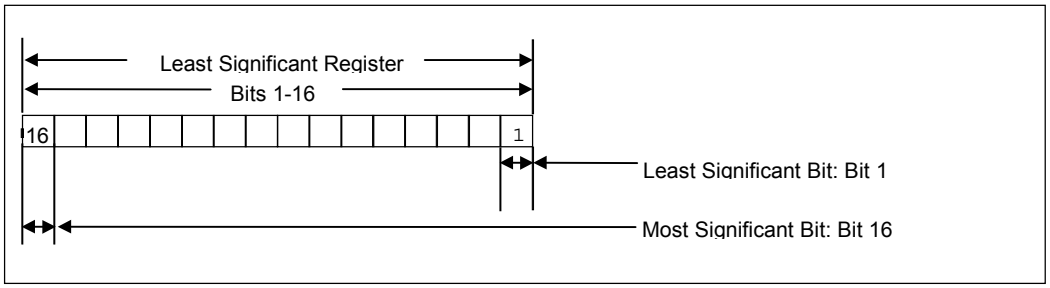
Precision – Related to the number of significant digits that can be stored. Since a double integer register uses 31 bits to store a number (bit 32 is used for the sign), it can potentially store numbers with greater precision than a real (floating-point) register, which only uses 23 bits to store a number's mantissa.

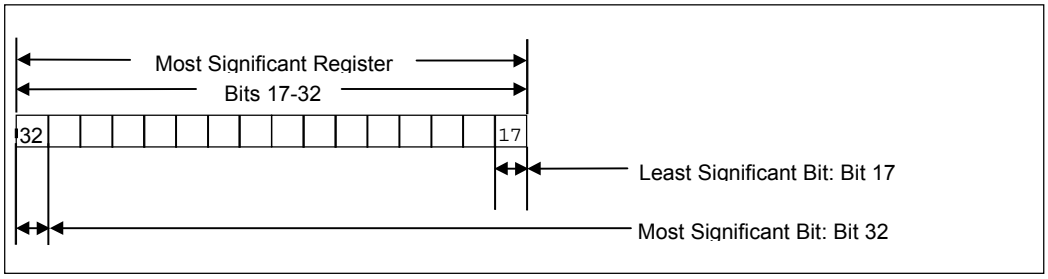
Internal Format of Floating-Point Numbers

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.



Register use by a single floating-point number is diagrammed below. In this diagram, if the floating-point number occupies registers %R0005 and %R0006, for example, %R0005 is the least significant register and %R0006 is the most significant register.





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. 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 programming 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 in Logicmaster
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 or incorrect floating-point number entries are shown below.

Incorrect Entry	Explanation/Result
–433E23	Missing decimal point. LM90 displays message “Bad numeric value.”
10e-19	Missing decimal point. LM90 displays message “Bad numeric value.”
1 0.e19	There is a space between the 1 and the 0 in the mantissa. Real numbers must be entered without spaces between digits or characters. Logicmaster recognizes this entry as the incorrect value +1.000000.
4.1e 19	There is a space between the e and the 19 in the exponent. Real numbers must be entered without spaces between digits or characters. Logicmaster recognizes this entry as the incorrect value +4.100000.

Errors in Floating-Point Numbers and Operations

Positive and Negative Infinity

On a 352 or 374 CPU, overflow occurs when a number greater than 3.402823E+38 or less than -3.402823E+38 is generated by a REAL function. On all other 90-30 models that support floating point operations, the range is greater than 2^{16} or less than $-2^{16.}$ When your number exceeds the range, the ok output of the function is set OFF, and the result is set to positive infinity (for a number greater than 3.402823E+38 on a 352 or 374 CPU or 2^{16} on all other models) or negative infinity (for a number less than –3.402823E+38 or -2^{16} on all other models). You can determine where this occurs by testing the sense of the ok output.

Mnemonic	Ladder Screen Value	Reference Table Value (Hex)	Description
POS_INF	+OVERFLOW	7F80 0000	IEEE positive infinity representation in hex.
NEG_INF	-OVERFLOW	FF80 0000	IEEE negative infinity representation in hex.

Note

If you are using software floating point (all models capable of floating point operations except the 352 or 374 CPU), numbers are rounded to zero (0) at ±1.175494E–38.

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.

Not a Number (NaN)

A Not a Number is an undefined number such as the result of dividing zero by zero. Positive and Negative Infinities are not considered to be NaNs. The following sections will help you identify when an NaN result has been obtained.

NaN Codes for 352 or 374 CPU

On a 352 or 374 CPU, each REAL function capable of producing an NaN produces a specialized NaN code that identifies the function and can be read in the applicable Reference Table. The indication on the Logicmaster ladder logic screen will be the unsigned term "OVERFLOW." (If the term "OVERFLOW" is preceded by a plus or minus sign, it indicates a positive or negative infinity.)

Not a Number (NaN) Codes for the 352 and 374 CPU		
Mnemonic	**Reference Table Value (Hex)**	**Description**
NaN_ADD.	7F81 FFFF	Real addition error value in hex.
NaN_SUB	7F81 FFFF	Real subtraction error value in hex.
NaN_MUL	7F82 FFFF	Real multiplication error value in hex.
NaN_DIV	7F83 FFFF	Real division error value in hex.
NaN_SQRT	7F84 FFFF	Real square root error value in hex.
NaN_LOG	7F85 FFFF	Real logarithm error value in hex.
NaN_POW0	7F86 FFFF	Real exponent error value in hex.
NaN_SIN	7F87 FFFF	Real sine error value in hex.
NaN_COS	7F88 FFFF	Real cosine error value in hex.
NaN_TAN	7F89 FFFF	Real tangent error value in hex.
NaN_ASIN	7F8A FFFF	Real inverse sine error value in hex.
NaN_ACOS	7F8B FFFF	Real inverse cosine error value in hex.
NaN_BCD	7F8C FFFF	BCD-4 to real error.
REAL_INDEF	FFC0 0000	Real indefinite, divide 0 by 0 error.

NaN Code for 35x, 36x, and 37x CPUs (excluding 352 CPU)

All Series 90-30 CPUs that support firmware-based floating point operations (which excludes the 352 CPU, which is hardware-based) produce only one NaN output: FFFF FFFF. The indication on the Logicmaster ladder logic screen will be the unsigned term "OVERFLOW."

Not a Number (NaN) Type for 35x, 36x, and 37xCPUs (Excluding 352 CPU)		
Mnemonic	**Reference Table Value (Hex)**	**Description**
NaN_SW	FFFF FFFF	Software Floating Point code for all NaNs

Propagation and Power Flow for NaN and Infinity Numbers

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 ok output is OFF (not energized).

The following table explains when power is or is not passed when dealing with numbers viewed as or equal to infinity for binary operations such as Add, Multiply, etc. As shown previously, outputs that exceed the positive or negative limits are viewed as POS_INF or NEG_INF respectively.

Table E-1. General Case of Power Flow for Floating-Point Math Operations

Operation	Input 1	Input 2	Output	Power Flow
All	Number	Number	Positive or Negative Infinity	No
All Except Division	Infinity	Number	Infinity	Yes
All	Number	Infinity	Infinity	Yes
Division	Infinity	Number	Infinity	No
All	Number	Number	NaN	No

Appendix F Programming Software Comparison

This manual was written for users of Logicmaster (a DOS-based PLC programming software). The Windows-based PLC software products, such as CIMPLICITY® Machine Edition Logic Developer and VersaPro®, provide PLC instruction set information in the software's built-in on-line help system rather than in a manual. Users of the Windows-based programming software should be aware that instructions appear differently than the way they appear on a Logicmaster screen (they still work the same in the PLC). The online help system has the most accurate information about using the instruction set in the Windows-based programming software.

In addition to the on-line help system, you can refer to the following manuals for information on using the software:

VersaPro™ Programming Software User's Guide, GFK-1670

CIMPLICITY® Machine Edition Getting Started Guide, GFK-1868

Notes

Support for DRUM Sequencer Instruction

This instruction, supported by CPUs 350-364 release 10.00 and later, and all versions of CPU37x , is not supported in any version of Logicmaster; therefore, not discussed in this manual. This instruction is supported in VersaPro, starting with release 1.1, and in all versions of Logic Developer. Information for this instruction can be found in the on-line help built into these two software packages.

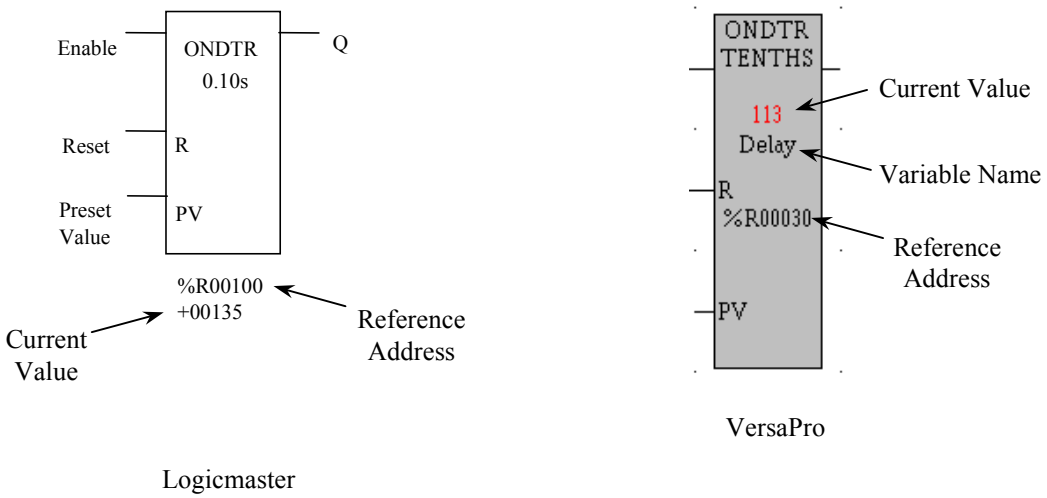
Start and End of Program Markers

These are used in Logicmaster ladder logic screens, but are not visible on the Windows-based programmers' ladder logic screens.

Instruction Control Word Address Location

Certain instructions, such as timers, counters, and the bit sequencer require a group of consecutive words to store certain internal calculations. This group of words is usually called a control block. In Logicmaster, the address of the first word of the control block (as well as any value stored in that address) appears below the instruction on the ladder logic screen (as %R00100 in the figure below). For the Windows-based programmers, this reference address of the first word appears inside the instruction on the ladder logic screen (as %R00030 in the figure below). VersaPro also displays

the Variable Name of the reference address (Delay in the in figure below) inside the instruction. If no one has assigned a Variable Name to the reference address, the address itself will be the default Variable Name (so the reference address would appear inside the instruction in both places). Right above the word Delay in the VersaPro view is the value 113, which represents the current value stored in that variable.



Real Number Display Differences

There are differences between the way the programs display undefined results such as when a divide by zero calculation is attempted. Appendix E of this manual discusses how Logicmaster displays these results in both the ladder screen and reference tables.

3

A

B

C

D

E

F

M

N

O

P

R

S

T

U