



GE Fanuc Automation

Programmable Control Products

Series 90™ Programmable Coprocesor Module and Support Software

User's Manual

GFK0255K

November 1999

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.

The following are trademarks of GE Fanuc Automation North America, Inc.

| | | | |
|-------------------|-------------|-------------|--------------|
| Alarm Master | Genius | ProLoop | Series Three |
| CIMPLICITY | Helpmate | PROMACRO | VersaMax |
| CIMPLICITY 90-ADS | Logicmaster | Series Five | VersaPro |
| CIMSTAR | Modelmaster | Series 90 | VuMaster |
| Field Control | Motion Mate | Series One | Workmaster |
| GENet | PowerTRAC | SeriesSix | |

Preface

The Programmable Coprocessor Module (PCM), from GE Fanuc Automation North America, Inc., is a high-performance microcomputer designed to perform coprocessor functions in a Series 90™ PLC system. It combines the function of the Communications Module (CCM) and the ASCII/BASIC Module (ABM), used on the Series Six™ programmable logic controller (PLC), into a single module with significantly greater capacity and performance than that of the ASCII/BASIC module.

Revisions to this Manual

Changes have been made to this version of the PCM manual, GFK-0255K, to add information about CPUs on page 3-5 (*Series 90-70 and Series 90-30 Minor Type codes*) and page 3-29 (*minimum system window time values*). Additionally, this manual describes configuration of the PCM using Logicismaster 90 configuration software; however, the PCM can also be configured with Control software. For information about Control software topics, refer to the Online Help for Control Programming software.

Content of this Manual

This manual contains the following chapters and appendixes:

Chapter 1. Introduction: describes the features of the PCM. System operation, module specifications, hardware features, and the use of various software tools are introduced in this chapter.

Chapter 2. Installing the PCM: explains how to install and configure the PCM in a Series 90-70 or 90-30 PLC system and how to install the necessary software.

Chapter 3. CCM Operation: describes CCM operation and features, and explains how to use the PCM for CCM applications.

Chapter 4. MegaBasic Operation: describes MegaBasic operation and features, and explains how to use the PCM for MegaBasic applications.

Chapter 5. Advanced MegaBasic Programming: describes the extensions to MegaBasic, as developed by GE Fanuc. These extensions allow MegaBasic to take full advantage of the special capabilities of the PCM and the Series 90 PLC system.

Chapter 6. Troubleshooting Guide: contains a self-guided demonstration of the steps involved in troubleshooting the PCM and application programs.

Appendix A. PCM Cabling Information: provides cabling specifications and wiring diagrams for the Series 90 PCM.

Appendix B. Resetting the PCM from a PLC Program: explains how COMMREQ function blocks may be used to reset the PCM.

Appendix C. PCM Commands: describes commands for loading, storing, and executing applications.

Appendix D. PCM Batch Files: describes how to create and use batch files.

Appendix E. Example MegaBasic Programs: provides a Megabasic test program.

Appendix F. TERMF File Descriptions: lists the files placed on the PCM programmer's hard disk during the INSTALL procedure.

Related PCM Publications

For more information on PCM, refer to these publications:

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

MegaBasic™ Programming Language Reference Manual (GFK-0256)

Programmable Coprocessor Module (PCM) Quick Reference Guide (GFK-0260)

PCM Development Software (PCOP) Quick Reference Guide (GFK-0657)

PCM Support Software (TERMF) Quick Reference Guide (GFK-0655)

Important Product Information for PCM Development Software (PCOP) (GFK-0352)

Important Product Information for PCM Support Software (TERMF) (GFK-0654)

Important Product Information for Series 90™ -70 PCM (GFK-0351).

Important Product Information for Series 90™ -30 PCM (GFK-0494)

Related Series 90 Publications

For more information on Series 90 programmable controllers, refer to these publications:

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

Logicmaster™ 90-70 Programming Software User's Manual (GFK-0263)

Series 90™ -70 Programmable Controller Reference Manual (GFK-0265)

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

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

Series 90™ -30/90-20 Programmable Controllers Reference Manual (GFK-0467)

Series 90™ PLC Serial Communications User's Manual (GFK-0582)

Series Six™ Data Communications Manual (GEK-25364)

Series 90™ -70 Programmable Controller User's Guide to Integration of Third Party VME Modules (GFK-0448)

Series 90™ -70 System Manual for Control Software Users (GEK-1192)

Control User's Manual (GEK-1295)

We Welcome Your Comments and Suggestions

At GE Fanuc Automation, we strive to produce quality technical documentation. After you have used this manual, please take a few moments to complete and return the Reader's Comment Card located on the next page.

Henry Konat
Technical Writer

| | | |
|------------------|---|-------------|
| Chapter 1 | Introduction | 1-1 |
| | Section 1: System Overview | 1-1 |
| | Section 2: Functional Overview | 1-2 |
| | CCM Operation | 1-2 |
| | MegaBasic Operation | 1-2 |
| | RAMDisk | 1-2 |
| | PCM Operation Modes | 1-3 |
| | PCM Support Utilities for Personal Computers | 1-3 |
| | Section 3: PCM Module Description | 1-4 |
| | LED Indicators | 1-4 |
| | OK LED | 1-5 |
| | User-Defined LEDs (USER1 and USER2) | 1-6 |
| | Battery | 1-6 |
| | Serial Connectors | 1-7 |
| | Section 4: Hardware Overview for the Series 90-70 PCM | 1-9 |
| | Section 5: Hardware Overview for the Series 90-30 PCM | 1-11 |
| | Section 6: Configuring the PCM | 1-13 |
| | Configuring the PCM for CCM Operation | 1-13 |
| | Configuring the PCM for MegaBasic Operation | 1-14 |
| | Section 7: Who Should Use PCOP | 1-15 |
| Chapter 2 | Installing the PCM | 2-1 |
| | What You Will Need | 2-2 |
| | Section 1: Installing the PCM Hardware | 2-3 |
| | Overview | 2-3 |
| | Installing a Series 90-70 PCM | 2-4 |
| | Installing a Series 90-30 PCM | 2-4 |
| | Adding Expansion Memory to the Series 90-70 PCM | 2-5 |
| | Section 2: Configuring the PCM with Logicmaster 90 Software .. | 2-6 |
| | I/OConfiguration Rack Screen | 2-6 |
| | Adding a PCM to the Rack Screen | 2-8 |
| | PCM Configuration Data | 2-9 |
| | PCM Configuration Modes | 2-10 |
| | Configuration Modes and the PCOP Display | 2-17 |
| | Series 90-30 PCM Autoconfig | 2-18 |

| | |
|---|-------------|
| Section 3: Configuring the Series 90-30 PCM with the HHP | 2-19 |
| Freezing the Configuration | 2-19 |
| Example of Editing a PCM | 2-20 |
| Section 4: TERMF Installation and Configuration | 2-24 |
| Installing TERMF | 2-24 |
| Adding the PCOP Directory to the MS-DOS Search Path | 2-25 |
| Section 5: Using TERMSET to Configure TERMF or PCOP | 2-26 |
| Local Configuration File | 2-31 |
| Connecting the PCM to the Programmer | 2-32 |
| Diagnosing Serial Communication Problems | 2-33 |
| Chapter 3 CCM Operation | 3-1 |
| Section 1: Series 90 CCM Target Memory Types | 3-2 |
| CCM Scratch Pad | 3-4 |
| Diagnostic Status Words | 3-6 |
| Section 2: Series 90 CCM Memory Addressing Conventions | 3-7 |
| Target/Source Memory Addresses | 3-7 |
| Data Length | 3-8 |
| CCM Comparisons | 3-9 |
| Section 3: Communications Request (COMMREQ) | 3-12 |
| Format of the COMMREQ Function Block | 3-13 |
| Other COMMREQ Faults | 3-15 |
| Power-Up Delay | 3-15 |
| Command Block | 3-15 |
| CCM Status Word | 3-18 |
| Section 4: CCM COMMREQ Data Block | 3-19 |
| Section 5: CCM COMMREQ Status Word | 3-23 |
| Section 6: CCM COMMREQ Example | 3-25 |
| Section 7: PLC System Communications Window | 3-28 |
| Series 90-70 System Communications Window | 3-28 |
| PLC Service Request (SVCREQ) | 3-28 |
| Series 90-30 System Communications Window | 3-30 |
| SVCREQ Examples | 3-30 |

| | | |
|------------------|---|-------------|
| Chapter 4 | MegaBasic | 4-1 |
| | Section 1: Programming the PCM in MegaBasic | 4-2 |
| | Getting Started with the MegaBasic Interpreter | 4-3 |
| | Loading and Saving MegaBasic Programs | 4-4 |
| | Backing Up Your Program | 4-5 |
| | Exiting the MegaBasic Interpreter | 4-6 |
| | Saving Data through a Power Cycle or Reset | 4-6 |
| | Compatibility with MS-DOS MegaBasic | 4-7 |
| | MegaBasic Features Not Supported by the PCM | 4-8 |
| | Modifying Existing BASIC Programs for MegaBasic | 4-8 |
| | Printing a MegaBasic Text File | 4-9 |
| | Using a Text Editor to Create MegaBasic Programs | 4-9 |
| | MegaBasic Program and Data Size | 4-9 |
| | Determining the Size of a MegaBasic Program | 4-10 |
| | MegaBasic Program Packages | 4-11 |
| | Changing the MegaBasic Workspace Size | 4-11 |
| | Compacting and Encrypting Programs | 4-12 |
| | Section 2: Interfacing to the PCM Hardware and Series 90 CPU | 4-13 |
| | Input and Output to the PCM Serial Ports | 4-14 |
| | Serial Port Control and Status | 4-14 |
| | Accessing PLC Data | 4-15 |
| | SYSLINK | 4-17 |
| | SYSREAD, SYSWRITE, and SYSTATUS\$ | 4-20 |
| | Status Record | 4-22 |
| | UNLINK Statement | 4-24 |
| | Data Coherency | 4-24 |
| | Accessing the PCM's LEDs | 4-25 |
| | Section 3: MegaBasic Programming Examples | 4-26 |
| Chapter 5 | Advanced MegaBasic Programming | 5-1 |
| | Section 1: MegaBasic Error Codes | 5-3 |
| | Section 2: Screen Formatting Commands | 5-5 |
| | CLS | 5-8 |
| | CUR\$ | 5-9 |
| | CUR | 5-10 |
| | ATTR\$ | 5-11 |
| | ATTR | 5-12 |
| | MV_CUR\$ | 5-13 |
| | MV_CUR | 5-15 |

| | |
|---|-------------|
| Section 3: Accessing %P, %L, and Password-Protected Data | 5-16 |
| CHG_PRIV | 5-17 |
| MSG_WTEXT | 5-18 |
| Section 4: Access to PLC Fault Tables and PLC Status | 5-21 |
| READ_FAULT_TBL | 5-23 |
| RDEL_FAULT_TBL | 5-24 |
| FLT_PRESENT% | 5-25 |
| FLT_CHANGED% | 5-26 |
| WORD% | 5-27 |
| Fault Table Header Records | 5-28 |
| PLC Fault Table Records | 5-28 |
| I/O Fault Table Records | 5-29 |
| Short Status Records | 5-30 |
| Time Stamp Subrecords | 5-32 |
| PLC Fault Address Subrecords | 5-32 |
| I/O Reference Address Subrecords | 5-32 |
| I/O Fault Address Subrecords | 5-33 |
| Known Problems with Fault Table Access | 5-33 |
| Section 5: Gathering PLC Information from MegaBasic Programs | 5-34 |
| READ_PLC_STATUS | 5-39 |
| READ_PLC_TIME_AND_DATE | 5-41 |
| READ_PLC_RUN_STATUS | 5-43 |
| READ_PLC_CPU_ID | 5-45 |
| CHECK_CPU_HEALTH | 5-46 |
| READ_PLC_FAULT_BIT | 5-48 |
| UTILITY_INIT | 5-49 |
| Section 6: Loading and Storing PCM Data Files Using TERMF . | 5-50 |
| L (Load) | 5-51 |
| S (Save) | 5-51 |
| D (file Directory) | 5-51 |
| X (eXterminate file) | 5-51 |
| Section 7: Serial Port Setup with IOCTL and PORT_CTL.BIN . | 5-52 |
| Serial Port Setup with IOCTL | 5-52 |
| Serial Port Control Using PORT_CTL.BIN | 5-55 |
| Section 8: Timers and Logical Interrupts | 5-57 |
| Timer Interrupts | 5-59 |
| Backplane Interrupts | 5-60 |

| | |
|--|--------------|
| Section 9: COMMREQs and Other Backplane Messages | 5-64 |
| PROCESS_MESSAGE Statement | 5-64 |
| Using the BKP_MSG Interrupt | 5-66 |
| Interpreting COMMREQ Messages | 5-67 |
| Programming the PLC COMMREQ Function Block | 5-68 |
| Format of the COMMREQ Instruction | 5-68 |
| MegaBasic COMMREQ Command Block | 5-71 |
| MegaBasic COMMREQ Example | 5-72 |
| MegaBasic Blink LED Program Example | 5-75 |
| Controlling COMMREQs | 5-76 |
| Identifying the Source of Backplane Messages | 5-82 |
| Backplane Messages to Another PCM | 5-86 |
| Section 10: Asynchronous Serial Input and Output | 5-91 |
| NOWAIT_OPEN | 5-92 |
| NOWAIT_READ and NOWAIT_WRITE | 5-94 |
| NOWAIT_CLOSE | 5-96 |
| NOWAIT_SEEK | 5-96 |
| NOWAIT_READ_ABORT | 5-97 |
| NOWAIT_WRITE_ABORT | 5-97 |
| Example NOWAIT Program | 5-98 |
| Section 11: VME Functions | 5-100 |
| VME Function Blocks for Communicating with the PCM | 5-100 |
| Some Rules for VME Bus Operations in Series 90-70 PLCs | 5-101 |
| General VME Information for the PCM | 5-101 |
| .PCM Dual Port RAM Available for Applications | 5-102 |
| VME Read Function | 5-103 |
| VME Write Function | 5-105 |
| VME Read/Modify/Write Function | 5-107 |
| VME Test and Set Function | 5-108 |
| MegaBasic Program Access to PCM Dual Port RAM | 5-109 |
| Section 12: Programming Example using VME Functions | 5-110 |
| Section 13: Optimizing Backplane Communication | 5-114 |
| Backplane Processing for the Series 90-70 PCM | 5-114 |
| Backplane Processing for the Series 90-30 PCM | 5-115 |
| Chapter 6 Troubleshooting Guide | 6-1 |

Contents

| | |
|--|------------|
| “OK” LED Not On | 6-1 |
| Reset Blinks User LED1 or LED2 | 6-1 |
| Communication Failure | 6-2 |
| PLC Fault Table Entries | 6-3 |
| Backplane Transfer Failure | 6-4 |
| Insufficient Memory Error | 6-4 |
| Loss of Characters/MegaBasic Tx/Rx Failure | 6-6 |
| CCM Data Tx/Rx Failure | 6-7 |
| Configuration Problems | 6-8 |
| Appendix A PCM Cabling Information | A-1 |
| Cable and Connector Specifications | A-1 |
| Serial Connectors | A-2 |
| Cabling | A-5 |
| RS-232 Cables | A-7 |
| RS-422/RS-485 Cables | A-10 |
| Appendix B Resetting the PLC from a PCM Program | B-1 |
| Appendix C PCM Commands | C-1 |
| Accessing the Command Interpreter | C-1 |
| Interactive Mode | C-2 |
| Command Format | C-2 |
| Notation Conventions | C-3 |
| Commands | C-3 |
| @ (execute a batch file) | C-4 |
| B (configure LEDs) | C-4 |
| C (Clear the PCM) | C-5 |
| D (file Directory) | C-5 |
| F (show Free memory) | C-5 |
| G (Get hardware ID) | C-6 |
| H (get PCM firmware revision number) | C-6 |
| I (Initialize device) | C-7 |
| J (format EEROM device) | C-9 |
| K (Kill a task) | C-9 |
| L (Load) | C-10 |
| M (create a memory Module) | C-11 |
| O (get LED configuration) | C-11 |
| P (request status data) | C-12 |
| Q (set protection level) | C-13 |
| R (Run) | C-14 |
| S (Save) | C-15 |
| U (reconfigure the PCM) | C-15 |
| V (Verify a file) | C-15 |
| W (Wait) | C-16 |
| X (eXterminate file) | C-16 |
| Y (set upper memory limit) | C-17 |

| | | |
|-------------------|--|------------|
| Appendix D | PCM Batch Files | D-1 |
| | Running Batch Files | D-1 |
| | PCMEEXEC.BAT Files | D-2 |
| | HARDEXEC.BAT Files | D-2 |
| | User-Installed PCMEEXEC.BAT and HARDEXEC.BAT Files | D-3 |
| Appendix E | Example MegaBasic Program | E-1 |
| Appendix F | TERMF File Descriptions | F-1 |
| Appendix G | Synchronous Serial Mode Operation | G-1 |
| | Port 1 Pin Assignments | G-1 |
| | Synchronous Operation Modes for Port 1 | G-2 |
| | Synchronous Mode PCMA3 (Port 1) Control Registers | G-3 |
| | NEC72001I/O Addresses | G-3 |
| | NEC72001 Synchronous Clock Source Selection (CR15) | G-4 |
| | For Further Information | G-4 |

Contents

| | |
|--|------|
| Figure 1-1. Series 90-70 PCM | 1-4 |
| Figure 1-2. Series 90-30 PCM | 1-5 |
| Figure 1-3. Series 90-70 PCM Hardware Block Diagram | 1-9 |
| Figure 1-4. Series 90-30 PCM Hardware Block Diagram | 1-11 |
| Figure 2-1. Series 90-70 PCM Configurations | 2-3 |
| Figure 3-1. Series One Jr. PLC vs Series 90 PLC | 3-9 |
| Figure 3-2. Series One PLC vs Series 90 PLC | 3-9 |
| Figure 3-3. Series One Plus PLC vs Series 90 PLC | 3-10 |
| Figure 3-4. Series Three PLC vs Series 90 PLC | 3-10 |
| Figure 3-5. Series Five PLC vs Series 90 PLC | 3-11 |
| Figure A-1. Serial Port Pin Assignments for the Series 90-70 PCM | A-2 |
| Figure A-2. Serial Port Pin Assignments for the Series 90-30 PCM | A-3 |
| Figure A-3. WYE Cable Connections for the Series 90-30 PCM | A-4 |
| Figure A-4. PCM to Workmaster Computer | A-5 |
| Figure A-5. PCM to PC-AT Personal Computer | A-5 |
| Figure A-6. PCM to Cimplicity Model W Computer | A-5 |
| Figure A-7. PCM to Workmaster II Computer or PS/2 Computer | A-6 |
| Figure A-8. PCM to PCM with Hardware Flow Control (RS-232 only) | A-7 |
| Figure A-9. CCM2 to PCM (RS-232 only) | A-7 |
| Figure A-10. PCM to OIT with Hardware Flow Control (RS-232 only) | A-8 |
| Figure A-11. PCM to OIT without Hardware Flow Control (RS-232 only) | A-8 |
| Figure A-12. PCM to a 5-Pin Device, Full Hardware Flow Control | A-9 |
| Figure A-13. PCM to a 5-Pin Device, No Flow Control or Hardware Flow Control | A-9 |
| Figure A-14. PCM to PCM without Hardware Flow Control (RS-422/RS-485) | A-11 |
| Figure A-15. CCM2 to PCM (RS-422/RS-485) | A-11 |
| Figure A-16. PCM to OIT without Hardware Flow Control (RS-422/RS-485) | A-12 |
| Figure A-17. PCM to Series One/Series Three DCA(RS-422/RS-485) | A-12 |
| Figure A-18. 2-Wire RS-422/RS-485 PCM Hookup | A-13 |
| Figure A-19. CCM2 or Host Computer to Multiple PCMs (Multidrop) | A-14 |
| Figure A-20. PCM or Host Computer to Multiple PCMs (4-Wire Multidrop) | A-15 |
| Figure B-1. Soft Reset | B-2 |
| Figure B-2. Hard Reset | B-3 |

| | |
|--|-------|
| Table 1-1. Series 90-70 PCM Expansion Memory and Accessories | 1-10 |
| Table 1-2. Series 90-30 PCM Features | 1-12 |
| Table 1-3. Series 90-30 PCM Accessories | 1-12 |
| Table 2-1. Expansion Memory Boards | 2-5 |
| Table 2-2. PCM Configuration Modes | 2-10 |
| Table 2-3. Logicmaster 90 Configuration Mode | 2-17 |
| Table 2-4. Autoconfig Default Configuration Values | 2-18 |
| Table 3-1. Memory Types Supported by Series 90 CCM | 3-2 |
| Table 3-2. Memory Types for the CCM Single Bit Write Function (6110) | 3-2 |
| Table 3-3. Series Six Memory Types NOT Supported by Series 90 CCM | 3-2 |
| Table 3-4. Series One Memory Types vs. Series 90 CCM Memory Types | 3-3 |
| Table 3-5. Series Five Memory Types vs. Series 90 CCM Memory Types | 3-3 |
| Table 3-6. Scratch Pad Memory Allocation | 3-4 |
| Table 3-7. CCM Diagnostic Status Word Definitions | 3-6 |
| Table 3-8. Target/SourceMemory Addresses | 3-7 |
| Table 3-9. Unit Lengths of Series 90 CCM Memory Types | 3-8 |
| Table 3-10. COMMREQ Data Block for CCM Commands | 3-20 |
| Table 3-11. COMMREQ Data Block for CCM Commands (Explanations for Table 3-10) | 3-21 |
| Table 3-12. Series Six CCM Commands NOT Supported by Series 90 CCM | 3-22 |
| Table 3-13. CCM Serial Port Secondary Error Codes (High Byte of Diagnostic Status Word 1) | 3-23 |
| Table 4-1. Default Program Workspace and RAM Disk Sizes | 4-10 |
| Table 5-1. MegaBasic Error Codes | 5-3 |
| Table 5-2. VT100.PGM Functions and Procedures | 5-6 |
| Table 5-3. Integer and String Constants | 5-7 |
| Table 5-4. Request and Access Codes for PLC Memory Types | 5-19 |
| Table 5-5. READ_FLTPGM Functions and Procedures | 5-21 |
| Table 5-6. READ_FLTPGM Shared Definitions and Constants | 5-22 |
| Table 5-7. UTILITY.PGM Package Procedures | 5-35 |
| Table 5-8. UTILITY.PGM Package Shared Constants and Variables | 5-36 |
| Table 5-9. PCM Interrupts and Associated Defaults | 5-58 |
| Table 5-10. Structure of Backplane Messages | 5-65 |
| Table 5-11. Backplane Message Fields | 5-65 |
| Table 5-12. Message Type, Rack, Slot, and ID Values | 5-83 |
| Table 5-13. PCM Address Allocation by Slot and Rack for Standard Non-Privileged Access - 39H ... | 5-102 |
| Table G-1. Port 1 Pin Assignments: RS-232 (Synchronous Serial Mode Operation) | G-1 |
| Table G-2. Port 1 Pin Assignments: RS-422 (Synchronous Serial Mode Operation) | G-2 |
| Table G-3. Port 1 Control Registers | G-3 |
| Table G-4. Serial Controller Port Assignments | G-3 |
| Table G-5. Synchronous Clock Source Selection | G-4 |

Chapter 1

Introduction

Section 1: System Overview

The Programmable Coprocessor Module (PCM), from GE Fanuc Automation North America, Inc., is a high-performance microcomputer designed to perform coprocessor functions in a Series 90™ PLC system. It combines the function of the Communications Module (CCM) and the ASCII/BASIC Module (ABM), used on the Series Six™ programmable logic controller (PLC), into a single module with significantly greater capacity and performance than that of the ASCII/BASIC module.

The PCM is closely coupled to the Series 90 PLC and may be configured to function as:

- One or two independent CCM ports.
- One CCM port and one MegaBasic application using one serial port.
- One MegaBasic application using one or both serial ports.

The PCM communicates with the PLC CPU over the backplane and can access user and system data using extensions to the MegaBasic language. No application support is required in the PLC CPU. Dual tasking allows the PCM to run a MegaBasic program and support a CCM communications channel at the same time.

For CCM and smaller MegaBasic applications, the PCM can use on-board memory. The Series 90-70 PCM has 128K bytes of memory; however, some Series 90-70 MegaBasic applications require a daughter board that can expand memory by 64K, 128K, 256K, or 512K bytes. The Series 90-30 PCM is available with various memory sizes to support different applications. All PCM memory is supported by a long-life lithium battery, located on the module.

Each PCM occupies a single slot in a Series 90 rack. Up to 63 PCMs may be installed in a single Series 90-70 PLC system to improve access to serial I/O devices and to access PLC memory. In the Series 90-30 PLC, the Model 331, or higher model CPU may have up to 4 PCMs in the main rack. The number of PCMs allowed in a rack may be restricted if the power consumed in the rack exceeds the rating of the power supply.

The PCM can be configured to run CCM and/or MegaBasic programs using Logicismaster 90 configuration software. Additional configuration capabilities are provided through the PCM development software package (PCOP) or by using PCM batch files.

Section 2: Functional Overview

CCM Operation

For CCM applications, each port behaves like an independent window into the PLC for access by other devices using the CCM protocol, such as industrial computers and color graphic terminals. The implementation of CCM on the PCM supports access to most user references. Many applications which accessed Series Six user references using CCM can now support the Series 90 PLC with little or no change.

Either port of the PCM can be configured in CCM **MASTER**, **SLAVE**, or **PEER** mode. In this capacity, the PCM acts much like a Series Six CCM module, transferring data between the PLC and an external device. Configuration of the port parameters is done in the Logicmaster 90 configuration package, using **CCM** mode. In **SLAVE** or **PEER** mode, an external CCM device, such as a computer, can request and send PLC CPU data to the PCM's ports. In **MASTER** or **PEER** mode, the application program in the PLC CPU can use the COMMREQ instruction to initiate data transfers to a CCM device attached to a PCM port.

MegaBasic Operation

As a programmable coprocessor, the PCM may be programmed with a powerful BASIC language interpreter called **MegaBasic** to perform data acquisition, data storage and retrieval, real-time computing, and operator interface functions. GE Fanuc has added **extensions** to MegaBasic to permit the reading and writing of user references in the PLC. These extensions also access program and system status information, permitting the development of operator interface programs to control and monitor processes.

MegaBasic programs may be developed off-line, using a standard text editor on a personal computer, or on-line with a terminal attached to the PCM programming port. A separate version of the MegaBasic interpreter also allows programs to be developed off-line in the MS-DOS environment. The MS-DOS version, however, does not support the PCM extensions to MegaBasic.

A MegaBasic program must be loaded to PCM RAM prior to being run on the PCM. If the program is saved to the PCM RAM Disk (RAM:), the PCM can be configured so that the MegaBasic program is automatically copied into the MegaBasic program execution RAM and executed by the MegaBasic interpreter upon power-up or reset.

RAM Disk

The RAM Disk is an area of RAM used to simulate a disk drive. It contains all memory not used by MegaBasic. Like any other storage medium (hard disk or floppy disk), the RAM Disk is used to store program and data files. RAM Disk files are preserved by a battery when the PLC is powered off.

Caution

If you store files to a RAM Disk, they will remain only as long as power is present or the battery is connected and working.

Application programming errors, battery failure, battery disconnection, and the removal of the daughter board from a Series 90-70 PCM have the potential to destroy files stored in RAM. All RAM files should be backed up to a more permanent storage medium (hard disk or floppy disk).

PCM Operation Modes

The PCM has a Restart/Reset pushbutton that is used to place the module in **RUN** mode or **PROGRAM** mode.

If the Restart/Reset pushbutton is pressed for less than 5 seconds, the PCM is placed in **RUN** mode. This reset is referred to as a **softreset**. CCM is started up on ports configured for CCM operation. If the PCM has been configured to run a MegaBasic program and that program has been saved to the PCM RAM Disk (RAM:), the program is automatically copied to the program workspace and executed by the MegaBasic interpreter. A power cycle also causes the PCM to go to **RUN** mode.

If the Restart/Reset pushbutton is pressed continuously for 10 seconds, the PCM performs a reset operation and enters **PROGRAM DEVELOPMENT** mode. CCM operation and MegaBasic programs are stopped. This reset is referred to as a **hardreset**. Program and configuration development are performed in this mode.

PCM Support Utilities for Personal Computers

TERMF, IC641SWP063, is a terminal emulation software package for personal computers. It is used to program MegaBasic programs on the PCM and transfer program files between the PCM and the personal computer. Setup of TERMF terminal configuration data is performed through a companion program called **TERMSET** (For more information on TERMF, refer to chapter 2, section 4, *TERMF Installation and Configuration*.)

PCOP, IC641SWP061, is a development system for the PCM, used for applications requiring configuration beyond that supplied by Logicmaster 90 software. PCOP provides functions for configuration, programming and running MegaBasic, loading and saving files, and other status and control functions. PCOP also supports folder and file maintenance.

TERMF and PCOP may be run directly from the MS-DOS prompt, or they may be accessed through the Logicmaster 90 main menu by selecting PCM development package (F3).

TERMF, PCOP, and Logicmaster 90 configuration software may be run on a Workmaster II, Workmaster, or Cimstar™ I industrial computer with a hard disk, or on an IBM-PC, PC-XT, PC-AT, IBM System/2, or 100% compatible personal computer with a hard disk, at least 640K of RAM, and DOS Version 3.0 or later. For information on choosing whether to use TERMF or PCOP, see section 7, *Who Should Use PCOP*.

IBM-PC and IBM System/2 are registered trademarks of International Business Machines Corporation.

Section 3: PCM Module Description

The Programmable Coprocessor Module has a battery and three LEDs. The Series 90-70 PCM has two serial port connectors, while the Series 90-30 PCM has a single connector supporting two serial ports. Both the Series 90-70 PCM and the Series 90-30 PCM have a Restart/Reset pushbutton.

LED Indicators

The three LED indicators, shown in the following figures, are mounted along the top front edge of the PCM.

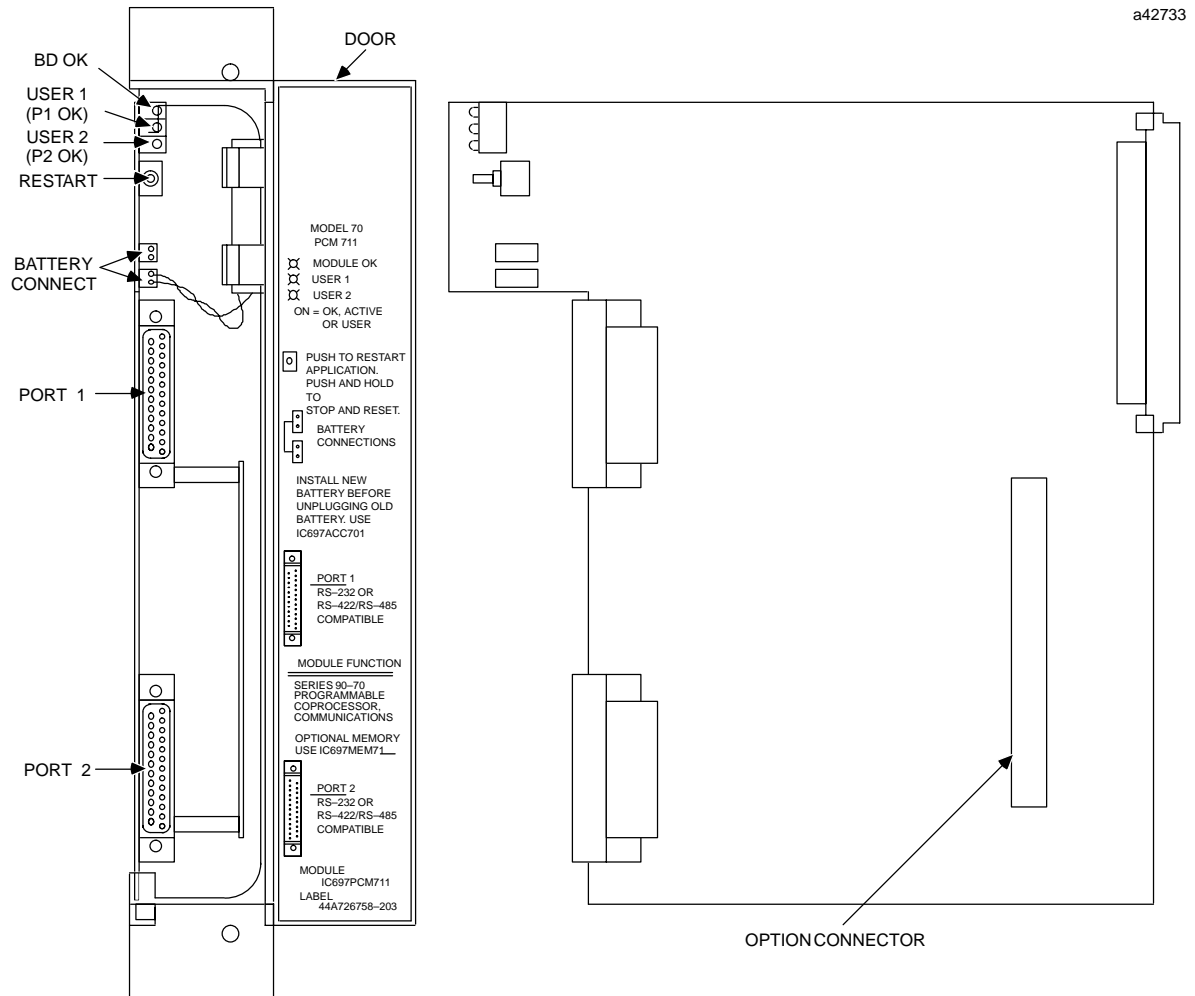


Figure 1-1. Series 90-70 PCM

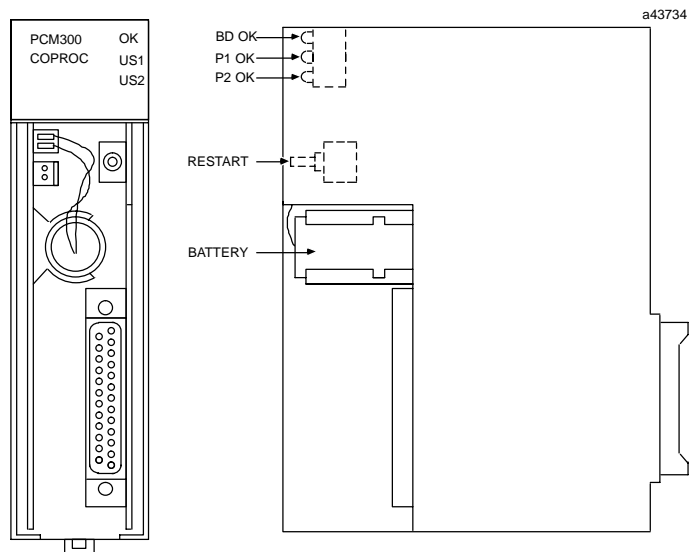


Figure 1-2. Series 90-30 PCM

OK LED

The OK LED indicates the current status of the PCM. It has three states:

| State | Description |
|----------|---|
| Off | When the LED is off, the PCM is <u>not</u> functioning. This is the result of a hardware malfunction, e.g., the diagnostic checks detect a failure, the PCM fails, or the PLC CPU is <u>not</u> present. Corrective action is required in order to get the PCM functioning again. |
| On | When the LED is on steadily, the PCM is functioning properly. Normally, this LED should <u>always</u> be on, indicating that the diagnostic tests were successfully completed and the configuration data for the module is good. |
| Flashing | The LED flashes during power-up diagnostics. |

Note

The PCM has a hardware watchdog timer that is periodically reset by the PCM software. If the watchdog timer expires, the PCM stops functioning and the OK LED turns off.

User-Defined LEDs (USER1 and USER2)

The remaining two LED indicators, USER1 and USER2, are user-definable LEDs. By default, these LEDs blink to indicate activity on the serial ports. USER1 blinks when port 1 sends or receives; USER2 blinks when port 2 sends or receives. The use of either or both user LEDs may be redefined.

Battery

A lithium battery is installed, as shown in figures 1-1 and 1-2. This battery maintains user memory when power is removed. Before the battery reaches the end of its useful life, a low battery fault is reported in the PLC fault table. See tables 1-1 and 1-3 for replacement battery catalog numbers.

When replacing a lithium battery, be sure to connect the new battery into the unused PCM battery connector before removing and discarding the old battery. Use the following procedure to replace the battery.

1. Open the front cover. Refer to figures 1-1 and 1-2 for the location of the battery on the module. (For a new PCM, the battery is not connected.)

Note

When the PCM is to be stored for an extended period of time, the battery should first be disconnected. However, if it is to be stored as a spare for a running application, you may wish to retain memory by leaving the battery connected.

2. Connect the battery to either battery connector on the module. If an old battery is present, connect the replacement battery to the unused battery connector before disconnecting the old battery. The tab on the connector should face to the right, away from the module surface.
3. Press down firmly to lock the battery connector in place, but do not force the battery connector into place.

Warning

Do not discard the lithium battery in a fire. Do not attempt to recharge the battery. Do not short the battery. The battery may burst, burn, or release hazardous materials. Manufacturer's instructions are available upon request.

Serial Connectors

The serial connectors on the PCM are used to communicate with external devices, such as operator interface terminals, bar code readers, and programming devices.

The Series 90-70 PCM has two serial connectors; each one supports both RS-232 and RS-485 operation. The serial ports are identical, and either port can be used for most applications. The two ports are configurable for different communication parameters.

Note

RS-485 is basically compatible with RS-422 devices.

Series 90-30 PCMs have a single serial connector that supports two ports. One port has a fixed interface. Port 1 uses RS-232 operation only. The 160K PCM, IC693PCM300 is restricted to using RS-485 on port 2. All other Series 90-30 PCM modules may select either RS-232 or RS-485 operation on port 2.

Caution

The serial ports on almost all versions of the PCM are connected to electrical ground within the PLC. Serial communication must be limited to distances of 50 feet (15 meters) unless ground isolation is provided by an external device. Failure to observe this caution may result in damage to the PCM or communicating device.

The serial ports on Series 90-70 PCMs using PCMA3 or newer hardware have limited isolation from electrical ground. These ports can withstand 200 volts DC plus instantaneous peak AC between any serial port connector pin (except shield) and PLC frame ground. This level of isolation is not adequate for many applications, for example, communication between devices on different power systems or where serial cables are exposed to intense electromagnetic fields. For applications like these, an external optical isolation device should be installed in the serial connection.

A WYE cable is supplied with each Series 90-30 PCM. The purpose of the WYE cable is to separate the two ports from a single physical connector; i.e., the cable separates the signals. In addition, the WYE cable makes cables used with the Series 90-70 PCM fully compatible with the Series 90-30 PCM.

The WYE cable is 1 foot in length and has a right-angle connector on one end that connects to the PCM. On the other end, it has a dual connector with one connector for port 1 and the other for port 2.

In order to use an RS-232 cable on port 2 of a Series 90-30 PCM, either a special cable must be made according to the serial port pin assignments shown in appendix A or a WYE cable must be used. Standard Series 90-70 PCM cables can be used for the Series 90-30 PCM when the WYE cable is used.

Caution

The WYE cable should not be used with Series 90-30 PCMs connected to an RS-485 multidrop network because it introduces signal reflections on the cable. Multidrop networks should be cabled directly to the PCM serial connector.

The connector pin assignments for the Series 90-70 PCM, Series 90-30 PCM, and the WYE cable shipped with each Series 90-30 PCM are shown in appendix A, *PCM Cabling Information*.

Section 4: Hardware Overview for the Series 90-70 PCM

The Series 90-70 PCM features include:

- A 12.5 mHz 80C186 microprocessor.
- 128 Kbytes on-board RAM.
- Up to 512 Kbytes optional RAM on memory expansion boards. These are the same expansion boards used by the Series 90-70 CPU 731/32 and CPU 771/772.
- Two RS-232/RS-485 serial ports.
- Backplane access to PLC memory.
- A real-time calendar clock synchronized to the PLC.
- A Restart/Reset pushbutton.
- Three status LEDs.
- Soft configuration (no dip switches or jumpers).
- Occupies a single slot in Series 90-70 racks.

Memory on the Series 90-70 PCM consists of PROM, local RAM, and dual port RAM. PROM contains the operating system, the CCM protocol, a built-in MegaBasic interpreter, and related utilities. Local RAM is divided into two data areas, one for PCM internal use and the remainder for the user's data and programs. Dual port RAM is used for communications between the PCM and the Series 90-70 PLC CPU.

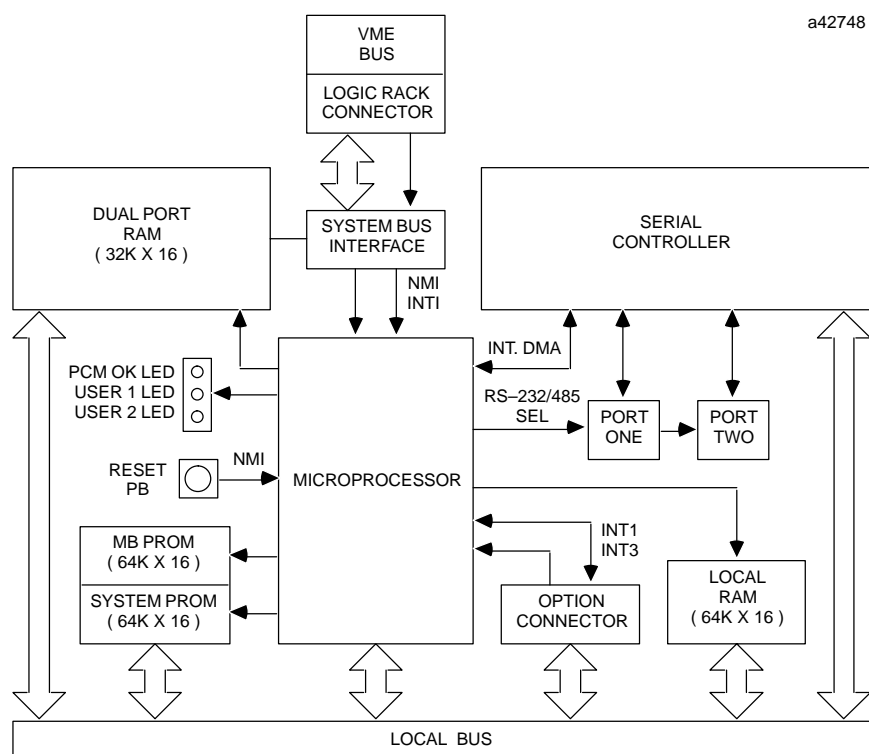


Figure 1-3. Series 90-70 PCM Hardware Block Diagram

Two serial ports are provided for communication with a programming terminal, CRTs, bar code readers, and other devices. These ports are identical in function. On the Series 90-70 PCM, both ports support RS-232 and RS-485 operation through software configuration.

The PCM has three LED indicators that enable you to determine the state of the PCM without having a terminal connected. The OK LED (top LED) indicates the current status of the PCM. The function of the USER1 and USER2 LED indicators (middle and bottom LEDs, respectively) can be configured by the user program or PCOP. By default, these LEDs indicate transmit and receive activity on serial ports 1 and 2, respectively.

The option connector on the Series 90-70 PCM provides for the addition of expansion memory. The Series 90-30 PCM does not have an option connector.

Table 1-1. Series 90-70 PCM Expansion Memory and Accessories

| Catalog No. | Description |
|--------------------|---|
| IC697MEM713 | Expansionmemory, 64 Kbytes. |
| IC697MEM715 | Expansionmemory, 128 Kbytes. |
| IC697MEM717 | Expansionmemory, 256 Kbytes. |
| IC697MEM719 | Expansionmemory, 512 Kbytes. |
| IC697ACC701 | Replacementbattery, package of 2. |
| IC690CBL701 | PCM to IBM XT compatible 9-pin serial connector. |
| IC690CBL702 | PCM to IBM AT compatible 9-pin serial connector. |
| IC690CBL705 | PCM to IBM PS/2 compatible 25-pin serial connector. |

Section 5: Hardware Overview for the Series 90-30 PCM

The Series 90-30 PCM features include:

- An 8 mHz 80C188 microprocessor.
- On-board RAM, options of 160, 192, or 640 Kbytes.
- Two serial ports, one RS-232 and one RS-232/RS-485.
- Backplane access to PLC memory.
- A real-time calendar clock synchronized to the PLC.
- A Restart/Reset pushbutton.
- Three status LEDs.
- Soft configuration (no dip switches or jumpers for most users).
- Occupies a single slot in the Series 90-30 I/O rack.

Memory on the Series 90-30 PCM consists of PROM and local RAM. PROM contains the operating system, the CCM protocol, a built-in MegaBasic interpreter, and related utilities. Local RAM is divided into two data areas, one for PCM internal use and the remainder for the user's data and programs.

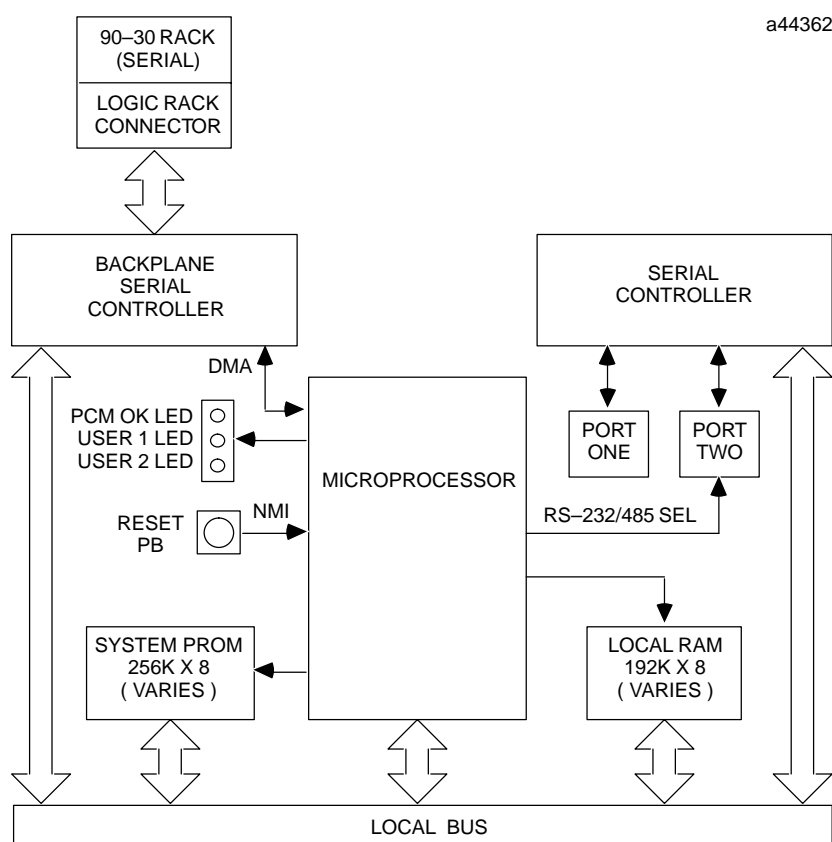


Figure 1-4. Series 90-30 PCM Hardware Block Diagram

Two serial ports are provided for communication with a programming terminal, CRTs, bar code readers, and other devices. These ports are identical in function and support RS-232 and/or RS-485/RS-422 operation through software configuration.

On the Series 90-30 PCM, connections to both ports are made through a single 25-pin D-type connector. A WYE cable is supplied with each PCM module. This cable is 1 foot in length and has a right-angle connector on one end that connects to the PCM. On the other end, it has a dual connector with one connector for port 1 and the other for port 2.

The PCM has three LED indicators that enable you to determine the state of the PCM without having a terminal connected. The OK LED (top LED) indicates the current status of the PCM. The function of the USER1 and USER2 LED indicators (middle and bottom LEDs, respectively) can be configured by Logicmaster 90 software, the user program, or PCOP. By default, these LEDs are used to indicate transmit and receive activity on serial ports 1 and 2, respectively.

Several versions of the Series 90-30 PCM are available. The best one for a particular application may be selected on the basis of memory and serial port requirements. The following table lists the PCM versions by catalog number, along with the total RAM memory available on each module.

Table 1-2. Series 90-30 PCM Features

| Module | Total RAM | Port 1 | Port 2 |
|---------------|------------------|---------------|------------------|
| IC693PCM300 | 160K | RS-232 | RS-485 |
| IC693PCM301 | 192K | RS-232 | RS-232 RS-485 |
| IC693PCM311 | 640K | RS-232 | RS-232 RS-485 |

Table 1-3. Series 90-30 PCM Accessories

| Catalog No. | Description |
|--------------------|---|
| IC693ACC301A | Replacement battery, package of 2. |
| IC693CBL305 | Series 90-30 PCM "WYE" cable. |
| IC690CBL701 | PCM to IBM XT compatible 9-pin serial connector. |
| IC690CBL702 | PCM to IBM AT compatible 9-pin serial connector. |
| IC690CBL705 | PCM to IBM PS/2 compatible 25-pin serial connector. |

Section 6: Configuring the PCM

Before a PCM can be used in a Series 90-70 or 90-30 PLC system, it must be configured for the CCM or MegaBasic functions it is to perform.

For some CCM-only applications, the Series 90-30 PCM requires no configuration at all. The Model 331, or higher model CPU provides an autoconfig setting for CCM to permit automatic operation. These autoconfig default settings for the Series 90-30 PCM in **CCM** mode are described in the next chapter.

Configuration information can be provided in two ways: through Logicmaster 90 configuration (or the Series 90-30 Hand-Held Programmer) and/or a user configuration created by the PCOP development software. For most MegaBasic and CCM applications, Logicmaster 90 configuration is all that is needed. Logicmaster 90 configuration of the PCM is described in section 2 of the next chapter.

When the PCM is configured for a MegaBasic application by Logicmaster 90 or the Series 90-30 Hand-Held Programmer, PCM batch files may be used to set certain PCM parameters. PCM batch files are described in appendix D of this manual.

For some advanced applications, the PCM may be configured using the PCOP development software. The configuration created by PCOP is called **UserConfiguration Data** (UCDF). The UCDF is similar in function to the CONFIG.SYS file in a DOS personal computer system. The PCM uses the information found in the UCDF to determine its function. For more information on creating user configuration data, refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487.

Configuring the PCM for CCM Operation

If the PCM is configured for **CCM ONLY** or **BAS/CCM** mode using the Logicmaster 90 configuration package, CCM operation is automatically started on the specified port(s) on power-up or following a soft reset. The configuration of port characteristics (data rate, etc.), unit ID, and assignment to port 1 or port 2 is also done with the Logicmaster 90 configuration package.

Configuring the PCM for MegaBasic Operation

If the PCM has firmware version 2.50 or greater, it may be configured for **BASIC** or **BAS/CCM** mode using Logicmaster 90 software. With these modes, the MegaBasic program in RAM: automatically loads and starts on power-up or following a soft reset (pressing the Restart/Reset pushbutton for less than 5 seconds). A hard reset (pressing the Restart/Reset pushbutton for 10 seconds) will cause MegaBasic to start in **PROGRAM DEVELOPMENT** mode. For more information on MegaBasic operation, see chapter 4, *MegaBasic*.

Note

PCMs with firmware versions 2.04 and earlier do not recognize **BASIC** or **BAS/CCM** mode. Using one of these modes will cause a fault to be posted in the PLC fault table for the PCM rack and slot location:
“Unsupported feature in configuration.”

Warning

The UCDF, PCM batch files, and MegaBasic program(s) in the PCM RAM: device are preserved during a PLC power loss by the PCM battery. The Logicmaster 90 configuration is preserved by the PLC CPU battery (Series 90-70) or power supply battery (Series 90-30). Failure to replace either of these batteries when necessary may prevent the PCM from operating normally.

Section 7: Who Should Use PCOP

Everyone who develops MegaBasic applications for the PCM will need to use either TERMF or PCOP. (See PCM Support Utilities in section 1 of this chapter for catalog numbers and brief descriptions of TERMF and PCOP.) TERMF provides a simple command line interface to the PCM command processor, while PCOP provides a menu-oriented interface similar to Logicmaster 90 software plus **EXPERT** mode shortcuts. PCOP also provides project folder support for multiple PCM applications.

Certain PCM configurations are available only from PCOP; users who need these configurations must use PCOP. Other users may prefer PCOP. We recommend PCOP for these users:

- CCM application developers who need to adjust CCM timeouts and turnaround times in finer increments than the ones provided by Logicmaster 90 configuration must configure the PCM with PCOP.

Note

Beginning with Release 3.00, timeouts and retry counts may be set using COMMREQs to the CCM task. Refer to the *Series 90 PLC Serial Communications User's Manual*, GFK-0582.

- Users who already use PCOP, are comfortable with it, and have existing User Configuration Data (UCDF) Files in their PCMs should continue to use PCOP. UCDFs can be created and stored to the PCM only by using PCOP.
- New users who are uncomfortable with the MS-DOS command line interface in personal computers will often be more productive using PCOP.
- Users who need the PCM command interpreter and file server active on serial port 2 during normal operation (i.e., while MegaBasic is using serial port 1) can set this up only with PCOP.
- Users of PCMs with firmware versions 2.04 and earlier must use PCOP to run MegaBasic. These users should consider upgrading firmware as an alternative. Contact the GE Fanuc PLC Technical Support Hotline at 1-800-GEFANUC for details.

If you decide to use PCOP, please refer to the *Series 90 PCM Development Software User's Manual*, GFK-0487, and skip section 4, *TERMF Installation and Configuration*, of chapter 2 in this manual.

Chapter 2

Installing the PCM

This chapter explains how to install a Programmable Coprocessor Module (PCM) in a Series 90 PLC system and how to install the necessary software. There are several easy steps in preparing the PCM for CCM communication or developing and executing a MegaBasic program.

This chapter is divided into the sections listed below. The necessary equipment and software packages required for the installation process are included in the following table. After that, each section describes one step of the installation procedure in detail.

Not all steps are necessary for every application.

| Section | Title | Description | Page |
|---------|--|--|------|
| 1 | Installing the PCM Hardware | Section 1 contains the first step in the installation procedure. This step is <u>always</u> required. It describes the physical installation of the PCM in a Series 90 rack. Hardware descriptions are also included. | 2-3 |
| 2 | Configuring the PCM with Logicmaster 90 Software | Section 2 contains the second step in the installation procedure. This step is <u>always</u> required for a Series 90-70 application and for most Series 90-30 applications (those <u>not</u> relying on autoconfig.) It describes how to add a PCM to the Series 90 I/O configuration, using Logicmaster 90 configuration software. | 2-6 |
| 3 | Configuring the Series 90-30 PCM with the HHP | The PCM parameters can be edited with the Hand-Held Programmer if you have a Release 3 or later CPU and a Release 2.51 or later PCM. Section 3 describes how to do this. | 2-19 |
| 4 | TERMFinstallation and Configuration | Section 4 describes how to install TERMF software on a programming computer and configure TERMF for the computer. This step is required for MegaBasic programming. | 2-24 |
| 5 | Using TERMSET to Configure TERMF or PCOP | Section 5 describes how to run TERMSET to modify the TERM.DAT file. If you are using PCOP, refer to the <i>Series 90 PCM Development Software (PCOP) User's Manual</i> , GFK-0487, for instructions on installing PCOP. | 2-26 |

What You Will Need

Before you can begin the installation procedure, you must have the following equipment:

- A Series 90-70 programmable controller (PLC) system or a Series 90-30 PLC system containing a model 331, or higher CPU.
- A Programmable Coprocessor Module (PCM) to install and test.

If your application is CCM with standard default settings for a Series 90-30 PCM, no other equipment or software is required.

For other applications, the following is also required:

- An MS-DOS based computer with a hard disk and MS-DOS version 3.0 or later. The computer may be:
 - A Workmaster II industrial computer.
 - An IBM PC-XT, PC-AT, industrial PC-AT, or PS/2 personal computer with an 83-key or 101-key keyboard.
 - A Workmaster or Cimstar™ I industrial computer with an 83-key or 101-key keyboard.
- Logicmaster 90 programming software.

For most CCM applications, no other equipment or software is required.

For MegaBasic and advanced CCM applications, you also need:

- PCM support software (TERMF), IC641SWP063, or PCM development software (PCOP), IC641SWP061.
- One of the standard RS-232 cables described in appendix A, *PCM Cabling Information*.

Section 1: Installing the PCM Hardware

The first step in the installation procedure is to physically install the PCM hardware and verify that it is working properly.

Overview

In a single rack system, the PCM resides in the same rack as the PLC CPU. In a multiple rack Series 90-70 PLC, the PCM can reside in either the CPU rack or an expansion rack. The Series 90-30 PCM must reside in the main rack with the CPU.

The following illustration shows two possible system configurations for installing a Series 90-70 PCM in either a local or expansion rack.

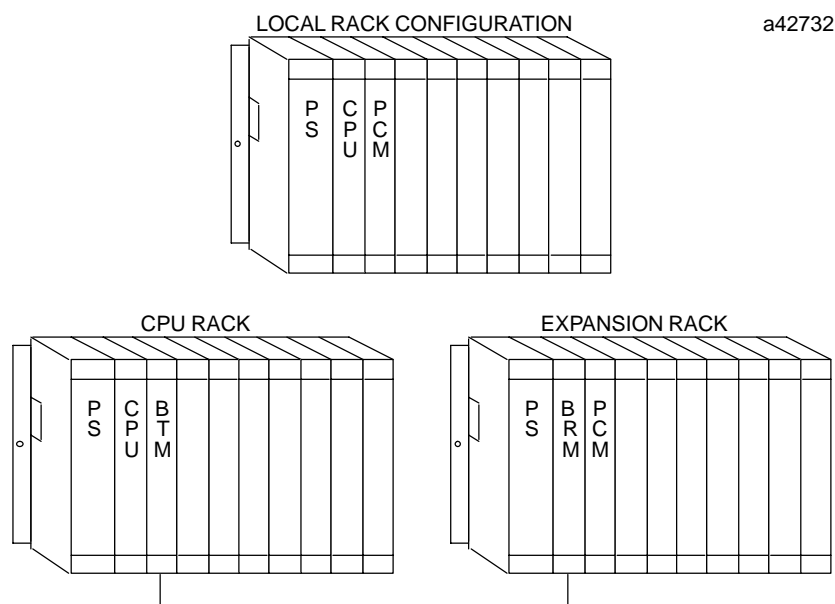


Figure 2-1. Series 90-70 PCM Configurations

The power supply, CPU, and Series 90-70 Bus Transmitter (BTM) or Bus Receiver Module (BRM) must reside in specific slots within each rack. The CPU module must be located in slot 1 of rack 0.

The Series 90-70 system usually includes a Bus Transmitter Module (BTM). The Bus Transmitter Module may be located in any slot as long as it is not to the right of an empty slot. If the PLC system has more than one rack, a Bus Receiver Module (BRM) must be located in slot 1 of each expansion rack.

Note

Version A of the Bus Transmitter Module must be installed to the right of all other GE Fanuc modules. There must be no empty slots between the Bus Transmitter Module and the CPU module.

The Programmable Coprocessor Module (PCM) may be placed in any unused slot in any rack, provided that these conditions are met:

- The configuration created by Logicmaster 90 configuration software must match the physical configuration of the modules. If it does not, the PLC may not operate as expected. Configuration faults are logged in the PLC fault table. Refer to the *Logicmaster 90 Programming Software User's Manual*, GFK-0263 or GFK-0466, for more information on PLC configuration using Logicmaster 90 software.
- When PCMs are installed in a Series 90-70 rack, all the slots between the PCM and the PLC CPU (CPU rack) or the Bus Receiver Module (expansion rack) must be occupied. If any of these slots is empty, the PCM cannot communicate across the backplane to the Series 90-70 PLC CPU or Bus Receiver Module.
- A Series 90-30 PCM must be in the main rack with the PLC CPU.

Installing a Series 90-70 PCM

To install a Series 90-70 PCM, follow these steps:

1. Set the CPU Run/Stop switch to STOP. This prevents the PLC program from initiating any command that may affect the operation of the module.
2. Power down the Series 90-70 PLC system.
3. Locate the desired rack and slot.
4. Remove the Series 90-70 PCM from the shipping carton, but leave it in its anti-static plastic bag. Touch an exposed metal surface of the PLC rack to discharge any electrostatic charge you may have picked up. Then remove the PCM from the protective bag.
5. Slide the PCM completely into the slot. The three LEDs are located at the top of the module.
6. Press the module firmly against the front rails of the PLC rack, but do not use excessive force.
7. Power up the PLC rack. The OK LED (top LED) on the faceplate flashes during power-up diagnostics. It continues to flash while waiting for configuration data from the CPU. If no signal is received across the backplane for 2 minutes, the PCM assumes that the CPU is not there and continue to power up without it. Once the PCM is ready, this LED should be continuously on.

Installing a Series 90-30 PCM

To install a Series 90-30 PCM, follow these steps:

1. Use Logicmaster 90 software or the Hand Held Programmer to stop the PLC. This prevents the PLC program from initiating any command that may affect the operation of the module.
2. Power down the Series 90-30 PLC system.
3. Locate the desired rack and slot.
4. Hook the top of the PCM case to the PLC baseplate and then guide the latch on the bottom of the case into its slot in the baseplate. The three LEDs are located at the top of the module.

5. Press down firmly to lock the module in place, but do not use excessive force.
6. Power up the PLC rack. The OK LED (top LED) on the faceplate of the PCM begins to flash during power-up diagnostics if a PLC CPU has been installed in the rack. When the PCM successfully completes diagnostics, it continues to flash while waiting for configuration data from the CPU. Once the PCM is ready, this LED should be continuously on.

Adding Expansion Memory to the Series 90-70 PCM

To increase the total available memory, an expansion memory daughter board may be added to a Series 90-70 PCM. The daughter board mounts on a single connector on the PCM. Expansion memory cannot be added to a Series 90-30 PCM.

Four versions of the expansion memory board are available:

Table 2-1. Expansion Memory Boards

| Catalog Number | Memory Size |
|----------------|-------------|
| IC697MEM713 | 64K Bytes |
| IC697MEM715 | 128K Bytes |
| IC697MEM717 | 256K Bytes |
| IC697MEM719 | 512K Bytes |

To mount the daughter board, follow these steps:

1. Touch an exposed metal surface of the PLC rack in order to discharge any electrostatic charge you may have.
2. Power down the Series 90-70 PLC system, and remove the PCM.
3. Remove the memory expansion module from its anti-static bag.
4. Carefully align the pins on the bottom side of the daughter board with the connector on the PCM.
5. Align the holes on the opposite end of the PCM with the daughter board standoffs.
6. Push the daughter board into the connector.
7. Make sure the daughter board is fully seated and the standoffs are snapped into both boards.

If an expansion memory board is installed on a PCM that has already been configured for no expansion memory or for a different expansion memory size, the Logicmaster 90 configuration must be updated and stored to the PLC CPU.

Section 2: Configuring the PCM with Logicmaster 90 Software

The second step in the PCM installation procedure is to add a PCM to the Series 90 I/O configuration, using Logicmaster 90 configuration software. The configuration software is used to describe the modules present in the PLC racks. Rack and slot location and other features for individual modules are specified by completing setup screens that represent the modules in a rack. Editing features make it easy to copy, move, replace, or delete module configurations.

After completing the Logicmaster 90 configuration, you must store it to the PLC where your PCM is installed. The configuration has no effect until it is stored to the PLC. The Logicmaster 90 status line must display **CONFIGEQUAL** after the configuration is stored to the PLC.

Note

For a Series 90-30 PCM using standard CCM default operation, this step is not required. Refer to the information on autoconfig at the end of this section for information on the default CCM operation.

I/O Configuration Rack Screen

From the main menu of the Logicmaster 90 configuration software, press I/O (F1). A screen representing the modules in a rack is displayed.

This example represents a Series 90-70 PLC rack.

| RACK | COPY | REF UU | DELETE | UNDEL | CFGSEL | 7une | 8other | 9 | 10zoom |
|---------|---------|--------|--------|---------|--------|------|--------|---|--------|
| 1n70 io | 2genius | 3bem | 4ps | 5rcksel | 6comm | | | | |

| PS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|--------|---------|---------|---------|---------|---------|---|---|---|---|
| PWR711 | CPU 731 | MDL 240 | PCM 711 | BEM 731 | ADC 701 | | | | |
| 100W | | I AC 16 | PCM | GBC1 | ADC | | | | |
| | | Ref Adr | 64 KB | Devices | | | | | |
| | | %100001 | | BUS1: 0 | | | | | |

| | | | |
|----------------|---------|-------------|--------------|
| D:\LM90\LESSON | OFFLINE | PRG: LESSON | CONFIG VALID |
| REPLACE | | | |

This example represents a Series 90-30 PLC Model 331 rack.

| RACK | COPY | REF UU | DELETE | UNDEL | | | | | | |
|---------|---------|--------|--------|---------|-------|---|--------|---|--------|--|
| 1m30 io | 2genius | 3 | 4ps | 5rcksel | 6comm | 7 | 8other | 9 | 10zoom | |

>

| PS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|--------|--------|---------|--------|--------|--------|-------|-------|-------|-------|-------|---|
| ===== | ===== | P | R | O | G | R | A | M | M | E | D |
| ===== | ===== | C | O | N | F | I | G | U | R | A | T |
| ===== | ===== | I | O | N | ===== | ===== | ===== | ===== | ===== | ===== | |
| PWR321 | CPU331 | MDL240 | QI 32 | APU300 | PCM300 | | | | | | |
| | | I AC16 | | HSC | PCM | | | | | | |
| | | RefAdr | RefAdr | | | | | | | | |
| | | z:10001 | QI0017 | | | | | | | | |

OFFLINE
C:\LM90\LESSON PRG: LESSON CONFIG VALID
REPLACE

The current module on a Series 90 I/O Configuration rack screen is highlighted in reverse video. Upon entering the rack screen, the Power Supply module is shown in reverse video. Use the left or right cursor keys to move from module to module. Use the up and down cursor keys to move between racks in descending or ascending order, respectively.

The rack screen presents an overview perspective of one Series 90 PLC rack. To add a PCM, press Other (F8) and then PCM (F1). To display the current configuration of the module in a slot, press Zoom (F10).

CONFIGVALID is displayed in the lower right corner of each display screen after the configuration is successfully validated. When **CONFIGINVALID** is indicated, the configuration may not be stored to the PLC. A **CONFIGINVALID** status is most likely to occur when:

- A slot in a Series 90-70 rack to the left of a module that generates interrupts, such as the PCM, is vacant.
- Input reference addresses (%I and %AI) overlap.
- Bus Transmitter and/or Receiver Modules in a multi-rack Series 90-70 PLC are missing.
- A Series 90-70 Version A BTM module is not located to the right of all other modules.

Adding a PCM to the Rack Screen

A PCM must first be entered before its memory expansion board may be selected. To do this, follow these steps:

1. Press Other (F8) and then PCM (F1) from the I/O Rack Configuration screen.

The screenshot shows the 'I/O Rack Configuration' screen. At the top, a rack bar displays slots 1 through 10. Slot 1 is labeled 'pcm', slot 2 is 'hsc', slot 4 is 'di', and slot 9 is 'expbd'. Below the rack bar, the text 'SERIES 90-70 MODULE IN RACK 0 SLOT 3' is displayed. A 'SOFTWARE CONFIGURATION' section is shown with a table for Slot 3. The table has three columns: 'CATALOG #', 'DESCRIPTION', and 'TYPE'. The first row shows '1', 'IC697PCM711', and 'PROGRAMMABLE COPROCESSOR MDL'. Below the table, instructions are given: '<< CURSOR TO THE DESIRED CATALOG NUMBER AND PRESS THE ENTER KEY >>' and '<< PRESS PGDN KEY FOR NEXT PAGE, PGUP KEY FOR PREVIOUS PAGE >>'. At the bottom, status indicators show 'OFFLINE', 'D:\LM90\LESSON', 'PRG: LESSON', and 'CONFIG VALID'.

| CATALOG # | DESCRIPTION | TYPE |
|-----------|-------------|------------------------------|
| 1 | IC697PCM711 | PROGRAMMABLE COPROCESSOR MDL |

2. Press the Enter key to enter the catalog number shown in reverse video and display the PCM detail screen.

The screenshot shows the 'PCM detail screen' for the 'IC697PCM711' module. The rack bar at the top is the same as in the previous screen. The 'SOFTWARE CONFIGURATION' section now shows 'Catalog #: IC697PCM711' and 'PROGRAMMABLE COPROCESSOR MDL'. Below this, a 'HELP (ALT-H) for Serial Port Restrictions' section is displayed. It contains two columns of configuration parameters for Port 1 and Port 2. The parameters include 'Config Mode', 'Battery Req', 'CCM Enable', 'CCM Mode', 'Interface', 'Data Rate', 'Flow Control', 'Parity', 'Retry Count', 'Timeout', 'TurnA Delay', and 'CPU ID'. At the bottom, status indicators show 'OFFLINE', 'D:\LM90\LESSON', 'PRG: LESSON', and 'CONFIG VALID'.

| Config Mode | Port 1 | Port 2 |
|------------------|---------------------|---------------------|
| CCM ONLY | CCM Enable : YES | CCM Enable : YES |
| Battery Req: YES | CCM Mode : SLAVE | CCM Mode : SLAVE |
| | Interface : RS232 | Interface : RS232 |
| | Data Rate : 19200 | Data Rate : 19200 |
| | Flow Control: NONE | Flow Control: NONE |
| | Parity : ODD | Parity : ODD |
| | Retry Count: NORMAL | Retry Count: NORMAL |
| | Timeout : LONG | Timeout : LONG |
| | TurnA Delay: NONE | TurnA Delay: NONE |
| | CPU ID : 1 | CPU ID : 1 |

- To enter a Memory Expansion board (for Series 90-70 systems only), press Expansion Board (F9) and then Memory (F1). The following list is displayed.

SERIES 90-70 MODULE IN RACK 0 SLOT 3

SOFTWARE CONFIGURATION

SLOT 3 Catalog #:

| CATALOG # | DESCRIPTION | TYPE |
|-----------|--|--------|
| 1 | IC697MEM713 MEMORY 64 KB CMOS EXPANSION | EXPMEM |
| 2 | IC697MEM715 MEMORY 128 KB CMOS EXPANSION | EXPMEM |
| 3 | IC697MEM717 MEMORY 256 KB CMOS EXPANSION | EXPMEM |
| 4 | IC697MEM719 MEMORY 512 KB CMOS EXPANSION | EXPMEM |

<< CURSOR TO THE DESIRED CATALOG NUMBER AND PRESS THE ENTER KEY >>
 << PRESS PGDN KEY FOR NEXT PAGE, PGUP KEY FOR PREVIOUS PAGE >>

OFFLINE
 D:\LM90\LESSON PRG: LESSON CONFIG VALID
 REPLACE

- Cursor to the desired catalog number, and press the Enter key.
- After selecting the desired memory expansion board, press the Escape key to return to the PCM detail screen.

PCM Configuration Data

After selecting the module and the expansion memory, the PCM is configured for CCM and/or MegaBasic operation from the PCM module detail screen. The PCM configuration mode is selected from the **Config Mode** field on the detail screen.

To select a different configuration mode, repeatedly press the Tab key until the desired mode is displayed on the screen. Then, press the Enter key. Use the Cursor and Tab keys to complete the detail screen for the desired configuration mode, and press the Enter key.

Once the PCM has been configured, press the Escape key again to save the module configuration and return to the rack display. The display screen should now include the PCM in the slot you selected. Complete the configuration procedure for each module you wish to add to the Series 90 PLC rack.

Once the Series 90 system configuration is complete, Logicmaster 90 software is used to store the configuration to the PLC CPU. If the PCM configuration has changed since the last configuration store, the PCM is automatically reset and restarts using the new configuration.

PCM Configuration Modes

Before configuring the PCM with Logicmaster 90 software, it is necessary to understand the various configuration modes and how they affect the operation of the PCM. Each mode is described in the table below. Many PCM applications use one of three configuration modes: CCM ONLY, BASIC, or BAS/CCM. Some applications use PCM CFG, PROG PRT, PROG/CCM, or CCM/PROG. These modes all require that the PCM itself be configured using the PCOP software package.

Table 2-2. PCM Configuration Modes

| Mode | Description |
|----------------------|--|
| CCM ONLY | CCM ONLY mode permits complete configuration for CCM execution on one or both PCM ports. After a soft reset, the PCM automatically starts CCM on the designated port(s). |
| BASIC | BASIC mode is used to configure the PCM for a standard MegaBasic application. The ports are configured for use by the MegaBasic program and for connection to the TERMF package (port 1), which is used to program the MegaBasic application. The program must be saved to the PCM RAM Disk (RAM:) as "BASIC.PGM". After a soft reset, the PCM automatically starts the MegaBasic program. BASIC mode is <u>not</u> available in Release 2 of Logicmaster 90-70 software. This mode is <u>not</u> supported by PCM firmware version 2.04 or earlier. |
| BAS/CCM | BAS/CCM mode is similar to BASIC mode, except that port 2 is configured for CCM execution and is, therefore, <u>not</u> available to the MegaBasic application program. After a soft reset, the PCM automatically starts the MegaBasic program, along with CCM on port 2. BAS/CCM mode is <u>not</u> available in Release 2 of Logicmaster 90-70 software. This mode is <u>not</u> supported by PCM firmware version 2.04 or earlier. |
| NONE | NONE is only used when the PCM does <u>not</u> support Logicmaster 90 configuration, as for a Release 1 PCM. NONE mode is only available in Logicmaster 90-70 software. |
| PCM CFG | PCM CFG indicates that all configuration data for the PCM is selected using PCOP and loaded to the PCM as User Configuration Data (UCDF) or that a PCMEEXEC.BAT file specifies the configuration to be used. |
| PROG PRT | PROG PRT is a special mode used to configure port settings for the PCM without automatically running a MegaBasic program or CCM. PROG PRT mode is selected for special applications using PCOP for configuration. The serial port configuration specified with this mode becomes effective only after a hard reset or if no other configuration exists on the PCM. |
| PROG/CCM CCM/PROG | PROG/CCM and CCM/PROG modes are a combination of CCM on one port and PCOP program part settings on the other port. |

Note

If User Configuration Data (UCDF) has been loaded to the PCM from PCOP, the UCDF configuration is used after a power cycle or soft reset. The Logicmaster 90 configuration is ignored. The PCOP **CLEAR** command or **MDE (Module Delete)** command can be used to delete the UCDF from the PCM.

CCM ONLY Mode

When CCM ONLY mode is selected, the PCM is used only for CCM operation on one or both ports. When this mode is selected in Logicmaster 90 software, the following PCM detail screen is displayed.

```

RACK 1 2hsc 3 4oi 5 6 7 8other 9expbd 10
>
SERIES 90-70 MODULE IN RACK 3 SLOT 3
SOFTWARE CONFIGURATION
SLOT 3 Catalog #: IC697PCM711 PROGRAMMABLE COPROCESSOR MDL
PCM 711
HELP (ALT-H) for Serial Port Restrictions
PCM Config Mode: CCM ONLY
Battery Req: YES
64 KB

----- Port 1 -----
CCM Enable : YES
CCM Mode : SLAVE
Interface : RS232
Data Rate : 19200
Flow Control: NONE
Parity : ODD
Retry Count: NORMAL
Timeout : LONG
TurnA Delay: NONE
CPU ID : 1

----- Port 2 -----
CCM Enable : YES
CCM Mode : SLAVE
Interface : RS232
Data Rate : 19200
Flow Control: NONE
Parity : ODD
Retry Count: NORMAL
Timeout : LONG
TurnA Delay: NONE
CPU ID : 1

OFFLINE
D:\LM90\LESSON PRG: LESSON CONFIG VALID
REPLACE

```

| Parameter | Description |
|-------------------|---|
| ConfigurationMode | Set to CCM ONLY . |
| BatteryRequired | Specify whether a battery is required. Choices are YES* or NO . |
| CCMEnable | Specify whether the port is to be configured for use as a CCM port. Choices are YES* or NO . |
| CCMMode | This parameter displays the availability of ports for CCM access. Choices are SLAVE* , PEER , or MASTER . |
| Interface | The interface parameter for port 1 of a Series 90-30 PCM is RS-232 , port 2 of the 192K and 640K Series 90-30 modules may have an interface value of either RS-232* or RS-485 . The interface parameter is <u>not</u> used or displayed for the 160K module, IC693PCM300. RS-232* and RS-485 are valid for both ports of the Series 90-70 PCM. |
| Data Rate | Data rate (bits per second or bps) for the port. Choices are 300, 600, 1200, 2400, 4800, 9600, or 19200*. |
| Flow Control | Type of flow control to be used for the port. Choices are HARDWARE or NONE* . |
| Parity | Type of parity to be used for the port. Choices are NONE or ODD* . |
| Retry Count | Retry counts for CCM mode. Choices are NORMAL* or SHORT . |
| Timeout | Length of timeouts used for CCM on the port. Choices are LONG* , MEDIUM , SHORT , or NONE . |
| TurnaroundDelay | Turnaround delay time to be used for CCM on the port. Choices are NONE* , 10 ms, 100 ms, or 500 ms. |
| CPUID | Address of the port on a multi-drop network. This value is used to calculate the backoff delay upon an inquiry collision in peer mode. The range of values allowed in this field is 1* to 254. A different value must be selected for each CCM device on the network. |

* Default selection.

BASIC Mode

Warning

BASIC mode is not supported in PCM firmware version 2.04 or earlier. Attempting to use it will result in an “Unsupported feature in configuration” fault for the PCM in the PLC fault table.

When BASIC mode is selected, the PCM automatically runs a MegaBasic program named **BASIC.PGM**, if it exists. CCM operation is not started on either port. When this mode is selected in Logicmaster 90 software, the following PCM detail screen is displayed.

The screenshot shows the 'SOFTWARE CONFIGURATION' screen for a Series 90-70 module in Rack 3, Slot 3. The module is identified as PCM 711 with a catalog number IC697PCM711. The configuration mode is set to BASIC. The screen displays settings for two serial ports, both configured as RS232C with a data rate of 19200, hardware flow control, no parity, 1 stop bit, and 8 bits per character. The module has 64 KB of memory. At the bottom, the status is 'OFFLINE' and 'CONFIG VALID'.

```

RACK 1 2 3 4 5 6 7 8 9 10
 1pcn 2hsc 3 4di 5 6 7 8other 9exbpd 10
>
SERIES 90-70 MODULE IN RACK 3 SLOT 3
SOFTWARE CONFIGURATION
SLOT 3 Catalog #: IC697PCM711 PROGRAMMABLE COPROCESSOR MDL
PCM 711
PCM
64 KB
Config Mode: BASIC
----- Port 1 -----
Interface : RS232C
Data Rate : 19200
Flow Contrl: HARDWARE
Parity : NONE
Stop Bits : 1
Bits/Char : 8
----- Port 2 -----
Interface : RS232C
Data Rate : 19200
Flow Contrl: HARDWARE
Parity : NONE
Stop Bits : 1
Bits/Char : 8
HELP (ALT-H) for Serial Port Restrictions
D:\LM90\LESSON PRG: LESSON
REPLACE OFFLINE CONFIG VALID
  
```

| Parameter | Description |
|--------------------|---|
| ConfigurationMode | Set to BASIC . |
| Interface | The interface parameter for port 1 of a Series 90-30 PCM is RS-232 ; port 2 of the 192K and 640K Series 90-30 modules may have an interface value of either RS-232* or RS-485 . The interface parameter is <u>not</u> used or displayed for the 160K module, IC693PCM300. RS-232* and RS-485 are valid for both ports of the Series 90-70 PCM. |
| Data Rate | Data rate (bits per second or bps) for the port. Choices are 300, 600, 1200, 2400, 4800, 9600, or 19200*. The Series 90-70 PCM also supports 38400. |
| FlowControl | Type of flow control to be used for the port. Choices are HARDWARE* , SOFTWARE , or NONE . |
| Parity | Type of parity to be used for the port. Choices are NONE* , ODD , or EVEN . |
| Stop Bits | The number of stop bits for the part. Choices are 1* or 2. |
| Bits per Character | The number of bits per character for data transfer on the port. Choices are 7 or 8*. |

* Default selection.

BAS/CCM Mode

Warning

BAS/CCM mode is not supported in PCM firmware version 2.04 or earlier. Attempting to use it will result in an “Unsupported feature in configuration” fault for the PCM in the PLC fault table.

When BAS/CCM mode is selected, the PCM automatically runs a MegaBasic program named **BASIC.PGM**, if it exists, on port 1 and CCM on port 2. When this mode is selected in the Logicmaster 90 software, the following PCM detail screen is displayed.

| RACK | | 2hsc | | 3 | | 4oi | | 5 | | 6 | | 7 | | 8other | | 9expbd | | 10 | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------------------|---------------------|---|--|---|--|-----|--|---|--|--------------|--|---|--|--------|-------------|--------|--|----|--|----------------------|--------------------|-------------------|------------------|-------------------|------------------|------------------------|-------------------|---------------|-------------------|---------------|--------------------|---------------|--------------|--|---------------------|--|----------------|--|-------------------|--|------------|
| > | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SERIES 90-70 MODULE IN RACK 0 SLOT 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SOFTWARE CONFIGURATION | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SLOT 3 | | Catalog #: IC697PCM711 PROGRAMMABLE COPROCESSOR MDL | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PCM 711 | | HELP (ALT-H) for Serial Port Restrictions | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PCM | | Config Mode: BAS/CCM | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 KB | | <table border="0"> <tr> <td>--- Port 1 BASIC ---</td> <td>--- Port 2 CCM ---</td> </tr> <tr> <td>Interface : RS232</td> <td>CCM Enable : YES</td> </tr> <tr> <td>Data Rate : 19200</td> <td>CCM Mode : SLAVE</td> </tr> <tr> <td>Flow Control: HARDWARE</td> <td>Interface : RS232</td> </tr> <tr> <td>Parity : NONE</td> <td>Data Rate : 19200</td> </tr> <tr> <td>Stop Bits : 1</td> <td>Flow Control: NONE</td> </tr> <tr> <td>Bits/Char : 8</td> <td>Parity : ODD</td> </tr> <tr> <td></td> <td>Retry Count: NORMAL</td> </tr> <tr> <td></td> <td>Timeout : LONG</td> </tr> <tr> <td></td> <td>TurnA Delay: NONE</td> </tr> <tr> <td></td> <td>CPU ID : 1</td> </tr> </table> | | | | | | | | | | | | | | | | | | --- Port 1 BASIC --- | --- Port 2 CCM --- | Interface : RS232 | CCM Enable : YES | Data Rate : 19200 | CCM Mode : SLAVE | Flow Control: HARDWARE | Interface : RS232 | Parity : NONE | Data Rate : 19200 | Stop Bits : 1 | Flow Control: NONE | Bits/Char : 8 | Parity : ODD | | Retry Count: NORMAL | | Timeout : LONG | | TurnA Delay: NONE | | CPU ID : 1 |
| --- Port 1 BASIC --- | --- Port 2 CCM --- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Interface : RS232 | CCM Enable : YES | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Data Rate : 19200 | CCM Mode : SLAVE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Flow Control: HARDWARE | Interface : RS232 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Parity : NONE | Data Rate : 19200 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Stop Bits : 1 | Flow Control: NONE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Bits/Char : 8 | Parity : ODD | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Retry Count: NORMAL | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Timeout : LONG | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | TurnA Delay: NONE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | CPU ID : 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| D:\LM90\LESSON | | | | | | | | | | OFFLINE | | | | | PRG: LESSON | | | | | | | | | | | | | | | | | | | | | | | | | | |
| REPLACE | | | | | | | | | | CONFIG VALID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

The configuration mode is set to **BAS/CCM**. The remainder of configurable parameters are as described previously for BASIC mode on port 1 and CCM mode on port 2.

PCM CFG and NONE Modes

When PCM CFG or NONE mode is selected in Logicmaster 90 software, there are no other parameters on the detail screen to be set. These modes require configuration data stored in the PCM. Either a **PCMEEXEC.BAT** file or User Configuration Data (UCDF) may be used. For information on **PCMEEXEC.BAT** files, see appendix D, *PCM Batch Files*. UCDFs are created and stored to a PCM using the PCOP development software. For more information on PCOP and UCDFs, see the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487. When one of these modes is selected, the PCM uses its default serial port settings unless they have been overridden by the **PCMEEXEC.BAT** or UCDF configuration.

An example detail screen showing NONE mode is displayed below. The detail screen for PCM CFG mode is identical to this.

| RACK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | | | | | | | | |
|------|-----|---|-----|---|---|---|----|---|---|----|--|---|--|---|-------|---|-------|----|--|
| 1 | pcn | 2 | hsc | 3 | | 4 | di | 5 | | 6 | | 7 | | 8 | other | 9 | expbd | 10 | |

>

SERIES 90-70 MODULE IN RACK 0 SLOT 3

| SLOT | SOFTWARE CONFIGURATION |
|---------|---|
| 3 | Catalog #: IC697PCM711 PROGRAMMABLE COPROCESSOR MDL |
| PCM 711 | HELP (ALT-H) for Serial Port Restrictions |
| PCM | Config Mode: NONE |
| 64 KB | |

| | | | |
|----------------|---------|-------------|--------------|
| D:\LM90\LESSON | OFFLINE | PRG: LESSON | CONFIG VALID |
| REPLACE | | | |

PROG PRT Mode

PROG PRT mode is used to configure the PCM ports without automatically configuring the PCM for MegaBasic or CCM operation. This mode requires configuration data stored in the PCM. Either a **PCMEEXEC.BAT** file or User Configuration Data (UCDF) may be used. For information on **PCMEEXEC.BAT** files, see appendix D, *PCM Batch Files*. UCDFs are created and stored to a PCM using the PCOP development software. For more information on PCOP and UCDFs, see the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487. PROG PRT mode is similar to PCM CFG mode and NONE mode but permits specifying serial port settings which are different from the default. However, the Logicmaster 90 serial port settings will be overridden by the **PCMEEXEC.BAT** or UCDF configuration data, if any.

Typically, use of this mode is to insure that port 2 has the same setup when PCM is in factory mode (hard reset) as when PCM is in user mode (soft reset). This is useful when developing MegaBasic programs that use port 2. When this mode is selected in Logicmaster 90 software, the following PCM detail screen is displayed.

```

RACK 1 pcm 2 hsc 3 4 oi 5 6 7 8 other 9 expbd 10
>
SERIES 90-70 MODULE IN RACK 0 SLOT 3
SOFTWARE CONFIGURATION
Catalog #: IC697PCM711 PROGRAMMABLE COPROCESSOR MDL
HELP (ALT-H) for Serial Port Restrictions
Config Mode: PROG PRT
----- Port 1 -----
Interface : RS232
Data Rate : 19200
Flow Contrl: HARDWARE
Parity : NONE
Stop Bits : 1
Bits/Char : 8
----- Port 2 -----
Interface : RS232
Data Rate : 19200
Flow Contrl: HARDWARE
Parity : NONE
Stop Bits : 1
Bits/Char : 8
OFFLINE
D:\LM90\LESSON PRG: LESSON CONFIG VALID
REPLACE

```

| Parameter | Description |
|--------------------|---|
| ConfigurationMode | Set to PROG PRT . |
| Interface | The interface parameter for port 1 of a Series 90-30 PCM is RS-232 ; port 2 of the 192K and 640K Series 90-30 modules may have an interface value of either RS-232* or RS-485 . The interface parameter is <u>not</u> used or displayed for the 160K module, IC693PCM300. RS-232* and RS-485 are valid for both ports of the Series 90-70 PCM. |
| Data Rate | Data rate (bits per second or bps) for the port. Choices are 300, 600, 1200, 2400, 4800, 9600, or 19200*. The Series 90-70 PCM also supports 38400. |
| FlowControl | Type of flow control to be used for the port. Choices are HARDWARE* , SOFTWARE , or NONE . |
| Parity | Type of parity to be used for the port. Choices are NONE* , ODD , or EVEN . |
| Stop Bits | The number of stop bits for the target port. Choices are 1* or 2 . |
| Bits per Character | The number of bits per character for data transfer on the target port. Choices are 7 or 8* . |

* Default selection.

PROG/CCM and CCM/PROG Modes

PROG/CCM and CCM/PROG modes are used to configure the PCM programming and CCM ports without automatically configuring the PCM for MegaBasic operation. These modes require configuration data stored in the PCM. Either a **PCMEEXEC.BAT** file or User Configuration Data (UCDF) may be used. For information on **PCMEEXEC.BAT** files, see appendix D, *PCM Batch Files*. UCDFs are created and stored to a PCM using the PCOP development software. For more information on PCOP and UCDFs, see the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487.

When PROG/CCM this mode is selected in Logicmaster 90 software, the following PCM detail screen is displayed.

| RACK | | 1pcn | | 2hsc | | 3 | | 4oi | | 5 | | 6 | | 7 | | 8other | | 9expbd | | 10 | | | |
|---|--|------------------------------|--|---------------------|--|---|--|-----|--|-------------|--|------------------------------|--|--------------------|--|--------|--|--------|--|--------------|--|--|--|
| > | | | | | | | | | | | | | | | | | | | | | | | |
| SERIES 90-70 MODULE IN RACK 0 SLOT 3 | | | | | | | | | | | | | | | | | | | | | | | |
| SOFTWARE CONFIGURATION | | | | | | | | | | | | | | | | | | | | | | | |
| SLOT 3 | | Catalog #: IC697PCM711 | | | | | | | | | | PROGRAMMABLE COPROCESSOR MDL | | | | | | | | | | | |
| HELP (ALT-H) for Serial Port Restrictions | | | | | | | | | | | | | | | | | | | | | | | |
| PCM 711 | | Config Mode: PROG/CCM | | --- Port 1 PROG --- | | | | | | | | | | --- Port 2 CCM --- | | | | | | | | | |
| PCM | | Interface : RS232 | | | | | | | | | | CCM Enable : YES | | | | | | | | | | | |
| 64 KB | | Data Rate : 19200 | | | | | | | | | | CCM Mode : SLAVE | | | | | | | | | | | |
| | | Flow Control: HARDWARE | | | | | | | | | | Interface : RS232 | | | | | | | | | | | |
| | | Parity : NONE | | | | | | | | | | Data Rate : 19200 | | | | | | | | | | | |
| | | Stop Bits : 1 | | | | | | | | | | Flow Control: NONE | | | | | | | | | | | |
| | | Bits/Char : 8 | | | | | | | | | | Parity : ODD | | | | | | | | | | | |
| | | | | | | | | | | | | Retry Count: NORMAL | | | | | | | | | | | |
| | | | | | | | | | | | | Timeout : LONG | | | | | | | | | | | |
| | | | | | | | | | | | | TurnA Delay: NONE | | | | | | | | | | | |
| | | | | | | | | | | | | CPU ID : 1 | | | | | | | | | | | |
| OFFLINE | | | | | | | | | | | | | | | | | | | | | | | |
| D:\LM90\LESSON | | | | | | | | | | PRG: LESSON | | | | | | | | | | CONFIG VALID | | | |
| REPLACE | | | | | | | | | | | | | | | | | | | | | | | |

The configuration mode is set to **PROG/CCM**. The remainder of configurable parameters are as described previously for PROG PRT mode on port 1 and CCM ONLY mode on port 2.

The screen displayed for CCM/PROG mode is similar. It shows CCM port options in the Port 1 column and PROG options in the Port 2 column.

Configuration Modes and the PCOP Display

When the PCOP development software is running and connected to a PCM, it attempts to display the current PCM operating mode on its status line. The following table shows the modes displayed by PCOP for various combinations of Logicmaster 90 configuration modes, presence or absence of UCDF configurations, and reset conditions. A soft reset occurs when the PCM Reset/Restart pushbutton is pressed and held for less than 5 seconds. A hard reset occurs when the button is held for 10 seconds.

Table 2-3. Logicmaster 90 Configuration Mode

| Configuration Present | | Soft Reset | | Hard Reset | |
|-----------------------------------|------|---------------------------------------|------------------------|---------------------------------------|------------------------|
| Logicmaster 90 Configuration Mode | UCDF | Configuration Used | Mode Displayed by PCOP | Configuration Used | Mode Displayed by PCOP |
| NONE | None | Factory Default | FACTORY | Factory Default | FACTORY |
| PCM CFG | None | Factory Default | FACTORY | Factory Default | FACTORY |
| CCM ONLY | None | LM90 CCM Config | LM CFG | Factory Default | FACTORY |
| PROG PRT | None | Factory Default LM90 PROG Port Def | LM CFG | Factory Default LM90 PROG Port Def | FAC MOD |
| PROG/CCM | None | LM90 CCM Config | LM CFG | Factory Default LM90 PROG Port Def | FAC MOD |
| NONE | Yes | PCM User Config | USER | Factory Default | FACTORY |
| PCM CFG | Yes | PCM User Config | USER | Factory Default | FACTORY |
| CCM ONLY | Yes | PCM User Config | USER | Factory Default | FACTORY |
| PROG PRT | Yes | PCM User Config | USER | Factory Default LM90 PROG Port Def | FAC MOD |
| PROG/CCM CCM/PROG | Yes | PCM User Config | USER | Factory Default LM90 PROG Port Def | FAC MOD |

Note that BASIC and BAS/CCM modes have been omitted from the table. When MegaBasic is started by the Logicmaster 90 configuration, PCOP will display either **LM CFG** when it can communicate with the PCM or **NO COMM** when it cannot.

Series 90-30 PCM Autoconfig

The Series 90-30 PLC Model 331, or higher CPU provides automatic default configuration called **Autoconfig** for many of the system modules.

The configuration provided for the Series 90-30 PCM is similar to configuring the PCM in Logicmaster 90-30 software for CCM ONLY mode. Thus, if Logicmaster 90-30 software has not been used to configure the Model 331, or higher CPU system, the Series 90-30 PCM can still be plugged into the rack and will communicate to a CCM device.

In order to initiate the autoconfig default settings, either clear the PLC or delete the PCM slot from the Logicmaster 90-30 configuration and then read it with the Hand-Held Programmer.

The default values for autoconfig, listed in the following table, are the same as the default configuration values provided by the Logicmaster 90-30 software in CCM ONLY mode.

Table 2-4. Autoconfig Default Configuration Values

| Port 1 | | Port 2 | |
|-------------|--------|-------------|---------------|
| CCMEnable | Yes | CCMEnable | Yes |
| CCMMode | Slave | CCMMode | Slave |
| Interface | RS-232 | Interface | RS-422/RS-485 |
| Data Rate | 19200 | Data Rate | 19200 |
| FlowControl | None | FlowControl | None |
| Parity | Odd | Parity | Odd |
| Retry Count | Normal | Retry Count | Normal |
| Timeout | Long | Timeout | Long |
| TurnADelay | None | TurnADelay | None |
| CPUID | 1 | CPUID | 1 |

To change any of the default configuration values provided or to use a different mode for the Series 90-30 PCM, you must use Logicmaster 90-30 software, PCOP, or both to configure the PCM. To use PCOP, refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487.

Note

If you have Release 3.00 or later PLC CPU and Release 2.50 or later PCM, the Hand-Held Programmer (HHP) may also be used to configure the PCM. Refer to the section 3 for more information on using the Hand-Held Programmer.

Section 3: Configuring the Series 90-30 PCM with the HHP

Programmable Coprocessor Module parameters can be edited with the Hand-Held Programmer if you have a Release 3 or later CPU and a Release 2.51 or later PCM. The parameters are edited in exactly the same manner as for Intelligent I/O Modules described previously in this chapter.

Freezing the Configuration

Processing a change to the PCM's configuration takes 15 seconds or more. Processing multiple parameter changes simultaneously takes the same time as processing a change to a single parameter. Since changing several parameters at once is a common occurrence, changes to individual parameters are remembered by the module but are not processed and do not take effect until specifically commanded to do so.

When a PCM parameter is changed, an asterisk (*) will appear before the module name on the top line of the HHP screen. This indicates that the module's previous configuration has been "frozen", and that the module is not yet using the change(s) you have just made. You can continue editing, and this and all subsequent changes will be remembered by the module. However, if power is lost while a module's configuration is frozen, the changes (edits) you have made will be lost.

When the configuration for a module is frozen in this manner, you can tell the system that editing of all of the parameters is complete by pressing the WRITE and ENT keys. The edited changes are then processed all at once by the PCM and the asterisk will disappear from the display, indicating that the new values are being used by the PCM and have been saved in the PLC's non-volatile memory.

If you decide to abandon the changes that you have made so far, they can be discarded by pressing the CLR and ENT keys. If you do this, the configuration parameters will revert to the values they had before the configuration was frozen.

If you attempt to leave the current slot (either by pressing the , -, or # key) while the module's configuration is frozen, you will be prompted to indicate whether to use the new combination of values, discard the new values and return to the old configuration, or to continue editing the changes. If you attempt to change the HHP mode or go to RUN mode, the "FROZEN" error message will be displayed.

Note

Once changes have been made which are not being used by the module, you cannot leave the slot until the changes are saved or discarded.


Example of Editing a PCM

For this example, assume that a 192K PCM (IC693PCM301) module resides in slot 2 of the CPU rack and that the PLC was powered up with the CLR and M/T keys depressed (that is, the PLC was cleared). In our example, we want to change the mode from CCM only (the default) to PROGRAMMER PORT and to change the data rate for both ports to 9600 baud.

Initial display:


```
R0:02 PCM301 <S
VERSION:3.01
```

To view the mode parameter:

Press the  key:

```
R0:02 PCM301 <S
MODR:CCM ONLY
```

To view other possible modes,

Press the  key:

```
R0:02 PCM301 <S
MODE:PROGRAM PRT
```

Each time that you press the -/+ key, other modes will be displayed. When the desired mode is displayed (it will be blinking),

Press the  key:

```
R0:02*PCM301 <S
MODE:PROGRAM PRT
```

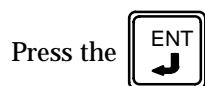
The asterisk to the left of PCM indicates that the module's configuration is now frozen. That is, the new mode value of PROGRAMMER PORT is remembered and displayed, but the module is still using the old value of CCM ONLY. If power were cycled at this time, the mode parameter would have the old value of CCM ONLY.

If you should attempt to change HHP modes or go to RUN mode when the module's configuration is frozen, the FROZEN error message will be displayed. For example:

Press the  key:

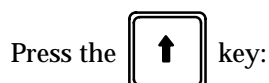
```
R0:08 FROZEN <S
MODE:PROGRAM PRT
```

To refresh the display of the module name, press any key, for example:



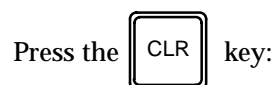
```
R0:02*PCM301 <S
MODE:PROGRAM PRT
```

If an attempt is made to view the configuration of a module in another slot at this time, the HHP will prompt you for the changes. For example:



```
SAVE CHANGES? <S
<ENT>=Y <CLR>=N
```

Since the port baud rate parameters have not yet been edited at this point in our example, we do not want to save the changes yet.



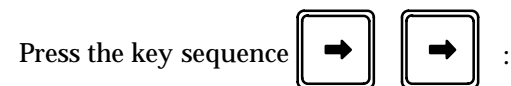
```
DISCARD CHGS? <S
<ENT>=Y <CLR>=N
```

If the changes are discarded at this time, we will lose the change we made to the mode parameter. That is, the configuration would revert to CCM ONLY, which is what it was before the configuration was frozen. Since we have more parameters to edit:



```
R0:02*PCM301 <S
MODE:PROGRAM PRT
```


Again, the asterisk indicates that the module's configuration is still frozen and the edited changes are not yet being used by the module. To display the baud rate parameter for port 1,



```
R0:02*PCM301 <S
DATA RT 1:19200
```

Notice that the asterisk remains to the left of the module's name. This indicates that the module's configuration is still frozen. It is possible to edit this and other parameters at this time, however none of the changes will be used by the module until they are saved as indicated below.

To change the port 1 baud rate to 9600:

Press the  key:


```
R0:02*PCM301 <S
DATA RT 1:9600
```

To display the baud rate parameter for port 2:

Press the  key six times:


```
R0:02*PCM301 <S
DATA RT 2:19200
```

To change the port 2 baud rate to 9600:

Press the  key:

```
R0:02*PCM301 <S
DATA RT 2:9600
```

To save the edited changes that we have made:

Press the  key:

```
SAVE CHANGES? <S
<ENT>=Y <CLR>=N
```

If the CLR key is pressed at this time, the SAVE operation will be aborted. Since we do want to save the changes,

Press the  key:

```
PROCESSING <S
CHANGES
```

The word PROCESSING will continue to blink until the module has completed processing of the new values. The HHP will then redisplay the last parameter that had been displayed:

```
R0:02 PCM301 <S
DATA RT 2:9600
```

Notice that the asterisk to the left of PCM301 is gone, indicating that the configuration is no longer frozen and that the module is using the new values.

To continue the example, suppose that you start changing parameters, then realize that you have made a mistake. The changes made so far (that is, since the configuration was frozen) can be discarded, reverting to the previous configuration.

Change the baud rate parameter for port 2 to 4800:

Press the   key sequence:

```
R0:02*PCM301 <S
DATA RT 2:4800
```

Notice that the configuration is frozen and that the actual baud rate being used by the PCM is 9600 (the previously configured baud rate).

To discard the changes,

Press the  key:

```
DISCARD CHGS? <S
<ENT>=Y <CLR>=N
```

If you press CLR again at this time, the discard operation would be aborted.

Press the  key:

```
R0:02 PCM301 <S
DATA RT 2:9600
```

The module's configuration is no longer frozen. The parameters have the same value they had before we changed the baud rate to 4800.

Section 4: TERMF Installation and Configuration

In order to use your personal computer to program the PCM, TERMF or PCOP must be installed and then configured. This section describes how to install TERMF on your computer and how to use TERMSET, the configuration program for TERMF and PCOP. For information on installing PCOP, refer to the *PCOP Quick Reference Guide*, GFK-0657, in the front of this manual, or to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487.

Installing TERMF

The PCM support software package, TERMF (IC641SWP063), includes one 5.25-inch and one 3.5-inch installation diskette. Choose the diskette that fits a diskette drive in your computer.

1. Select the TERMF installation diskette which fits your computer's A drive and place it in your computer. TERMF must be installed from drive A.
 - A. To install TERMF to hard drive C, type **a:install** at the MS-DOS prompt and press the Enter key.
 - B. If your computer has more than one hard drive or your hard drive is partitioned into two or more logical drives, you can install TERMF to hard drive D or E rather than C. To install TERMF to hard drive D, type **a:install d:** and press the Enter key at the MS-DOS prompt.
2. The TERMF installation program creates these directories on the specified hard drive if they do not already exist:

```
\PCOP
\PCOP\EXAMPLES.PCM
\PCOP\UTILS
```

The files listed in appendix F, *TERMF File Descriptions*, are copied to these directories. If a previous version of TERMF or PCOP was already on the hard drive before this installation, there may be additional files in these directories. Any files that are not listed in appendix F may be deleted from the hard disk in order to conserve storage space.

3. The CLEANUP batch file in the `\PCOP` directory may be used to delete unnecessary files. To use it, be sure that the hard drive where TERMF is installed is the current drive. Then, type the following at the MS-DOS prompt:

```
CD \PCOP
CLEANUP
```

Adding the PCOP Directory to the MS-DOS Search Path

The MS-DOS operating system in your computer uses a search path to find programs that are not in the current directory. The **PCOP** directory should be added to the search path so that **TERMF** and **TERMSET** can be used without typing the name of the directory where they are stored.

The MS-DOS **PATH** command may be used to determine what directories are currently in the search path. The **PATH** command can also be used to add the **PCOP** directory to the search path.

To determine what directories are currently in the search path, type **path** at the MS-DOS prompt and press the Enter key. MS-DOS will display a list of the directories, separated by semicolons, in the current search path on your computer's display. If no search path has been defined, the words **No Path** are displayed on the screen. If the **PCOP** directory is not already included in the search path, it should be added.

The search path is defined in the **AUTOEXEC.BAT** file, located in the root directory of the disk drive from which MS-DOS is started (the boot drive, which is usually drive C). Using any text editor program, edit the **\AUTOEXEC.BAT** file. If the file does not contain a path command similar to **path=c:\dos**, add the following command as the first line of the file:

```
path=c:\pcop
```

If there is already a **PATH** command, add this text at the end of the path definition:

```
;c:\pcop
```

If **TERMF** was not installed on the C drive, use the correct drive letter in place of "c".

If there is no **AUTOEXEC.BAT** file in the root directory of the boot drive, create one with your text editor. Include only a **PATH** command, as above, which specifies the **PCOP** directory on the correct hard drive.

Note

The **AUTOEXEC.BAT** file may begin with an **ECHO OFF** command in the first line. If one is present, the **PATH** command should appear in the second line.

Section 5: Using *TERMSET* to Configure *TERMF* or *PCOP*

Before *TERMF* or *PCOP* can be used to program the PCM, it requires information about the display hardware in your computer and the serial port settings to be used for communication with the PCM. This information is provided by a data file named **TERM.DAT**. A default version of **TERM.DAT**, installed with *TERMF* or *PCOP*, contains display hardware settings for the Workmaster industrial computer from GE Fanuc. These settings will work with any computer that uses an IBM Color Graphics Adapter (CGA) display. The default **TERM.DAT** also contains serial port settings identical to the PCM default settings.

If your computer uses different display hardware, such as the IBM Monochrome Display Adapter (MDA), Enhanced Graphics Adapter (EGA) or Video Graphics Array (VGA), you need to use the **TERMSET** program. If you change your computer's hardware configuration, you will need to use **TERMSET** again. You also need to run **TERMSET** in order to use PCM serial port settings that are different from the defaults. **TERMSET** asks you about your computer's display hardware and the serial port settings you plan to use; then it makes the necessary changes to the **TERM.DAT** file.

The file **DEFAULT.DAT** contains the same default settings as the initial version of **TERM.DAT** installed with *TERMF*.

Caution

The file *DEFAULT.DAT* should never be modified. It is very useful when troubleshooting for providing default settings to *TERMF* or *PCOP*.

To run **TERMSET**

1. Type **CD \PCOP** at the MS-DOS prompt.
2. To make changes to the **TERM.DAT** file, type **TERMSET** to display the **TERMSET** main menu. There are three parts to the main menu: Basic Setup, Custom Configurations, and Exit **TERMSET**. To configure *TERMF* or *PCOP* for your computer, you need to use only the two Basic Setup functions (items 3 and 4, below) to set up the serial port and display adapter. For information on Custom Configurations, see items 5 through 8 on the following pages.
3. To change the serial port settings, first type **1** to display the current values. Then type the number corresponding to the setting to be changed. A new menu will appear, showing all possible values for the selected setting. Choose one of the values from this menu. Be sure the new port settings match the configuration of the PCM port used for programming. When all the settings are correct, type **E** to return to the main menu.
4. To change the video adapter (display hardware) settings, type **2**. If the display settings are correct, answer **N** to the prompt to avoid changing them. Generally, the display adapter type is the only setting that should need to be changed. If it is incorrect, type **Y**. An explanation of the display parameters is provided in the following table.

| Value | Description |
|---------------------------|---|
| Video Adapter Type | Specifies the type of display adapter your personal computer uses to drive the attached video monitor (CRT). The six hardware selections are: 1 = Enhanced Graphics Adapter/Video Graphics Array (EGA/VGA). 2 = Color Graphics Adapter (CGA) (e.g. Workmaster computer). * 3 = Good CGA without snow (i.e. a non-IBM CGA-compatible adapter). 4 = Monochrome Display Adapter (MDA). 5 = Vega 7 Deluxe/Multisync (using VEGA BIOS.SYS). 6 = EGA driving a Monochrome Display. |
| Display Page Length | Specifies the number of lines to be displayed on one video page. Range = 10 - 60 lines (Default = 25). |
| Display Control Sequences | Display control character sequences from the PCM. Choices are ENABLED or DISABLED*. If display is enabled, control sequences are visible but have no effect. If display is disabled, control sequences perform their intended functions. |
| Normal Display Video Mode | Controls character width and color/monochrome features of the display. Range is normally 0 - 3 and 7. Recommended values are: 3 = Color display (video adapter type 1, 2, 3, or 5). * 7 = Monochrome display (video adapter type 4 or 6). |
| Display Long Lines | Specifies whether TERMF will honor the Set 132 column escape sequence, <ESC> ?3h. Choices are ENABLED or DISABLED*. |
| | It is recommended that you disable this feature unless you have a display adapter and driver software that interpret one of the two video modes listed above to display a 120 or 132-column screen. If you enable this selection, a second option allows you to specify which display mode represents your long-lines display mode. |
| Character Font | This selection is valid only for EGA/VGA adapters with the capability of using a user-loaded character font for text display. The four font selections are: 1 = Normal font in ROM*. This allows for 25 lines of text per screen. 2 = Double dot font in ROM. This allows 43 to 50 lines of text per screen. 3 = User specified 256-character font. This allows 10 to 60 lines of text per screen and up to 256 user-defined characters. 4 = User specified 512-character font. This allows 10 to 60 lines of text per screen and up to 512 user-defined characters in two font files of 256 characters each. The font sizes in the two files must be the same. |

* Default selection.

When all the display settings are correct, return to the main menu by typing **N** (or any key except **Y**) at the prompt that asks whether you want to change the settings. Then type **E** to save the changes and exit from **TERMSET**. If you do not want to save the new settings, type **Q**.

In addition to the settings previously described, some custom configuration options are available from the TERMSET main menu. These options are seldom, if ever, used with TERMF or PCOP. They are included here to help you configure TERMF as a general purpose terminal emulation program for use with serial devices other than the PCM. These options are explained in the remainder of this section.

5. Item 3 in the TERMSET main menu selects a screen that describes the keyboard input queue, special key handling, multi-character transmission delay, size of the received character buffer, key assignments for exiting from TERMF and for sending a break from the serial port, and the delay count which determines the length of the break. To change any of these parameters, type Y. A short explanation of each parameter is given below:

| Value | Description |
|------------------------------------|--|
| KeyboardInput Queue | Options include: <u>Supplied by BIOS*</u> : The capability to type ahead is limited to 15 characters. This mode is reliable on every computer. <u>It is strongly recommended</u> <u>Supplied by program</u> : The capability to type ahead is expanded to 255 characters. This mode is <u>not</u> reliable on all computers, and is <u>not</u> recommended. |
| Special Key Handling | Special keys include the function keys, ALT keys, and keypad keys, which have a "key binding" associated with them. <u>Sent as character sequences from key bindings</u> : When a key is pressed, the appropriate key binding sets are searched for that key. If found, the keystroke is translated accordingly. With TERMF in cursor mode, all three sets (cursor, application, and normal) are searched in that order. In application mode, only application and normal sets are searched; in normal mode, only the normal key binding set is searched. <u>Sent as MegaBasic control characters*</u> : When a key is pressed, a search is first performed in a predefined set of MegaBasic key bindings. (Refer to the <i>MegaBasic Programming Language Reference Manual</i> , GFK-0256.) The MegaBasic key bindings are designed to allow the same use of special keys for MegaBasic in both the PCM and a personal computer. If the keystroke does <u>not</u> match a MegaBasic key binding, translation is performed as described in the preceding option (sent as character sequences from key bindings). |
| Multi-character Transmission Delay | This delay value is arbitrary and varies among different computer models. It is a count of the number of times a software delay loop is repeated before putting the next character into the transmit buffer. This parameter is used only when TERMF is communicating with very slow devices. It is included here only for the sake of completeness. The PCM does <u>not</u> require this delay. Range = 0* to 32,767. <u>The value zero (0) is strongly recommended</u> |

* Default selection.

| Value | Description |
|-------------------------------|---|
| ReceivedCharacter Buffer Size | This buffer size affects the overall speed of TERMF when flow control is enabled. With no flow control, it affects the probability that a received character may be lost. It also affects the responsiveness of the display when a key is pressed: the smaller the buffer, the more responsive. If you use the Scroll Lock key to start and stop continuous output from a device, a 1000-character buffer is recommended. If you are simply starting and stopping a MegaBasic listing from the PCM using the space bar, then a 100-character buffer* is adequate. |
| Exit Term Key | <p>This selection changes the key combination which exits from TERMF. The default setting, 2092*, assigns ALT-Z. (Press the z key while holding the ALT key.) You can also exit from TERMF by using the CTRL-BREAK key combination.</p> <p>The scan codes used for this setting do <u>not</u> correspond to keyboardscan codes documented in IBM personal computer manuals. Use Show Scan Code, below, to determine the correct setting for the key or key combination (Shift, CTRL or ALT plus another key) you select.</p> <p>Do <u>not</u> change this setting if you use PCOP. The <i>Series 90 PCM Development Software (PCOP) User's Manual</i>, GFK-0487, specifies ALT-Z as the key combination to return to PCOP after communicating with PCM MegaBasic. If you change this setting, PCOP will <u>not</u> function as expected.</p> |
| Send Break Key | <p>This selection changes the key combination which causes TERMF to send a serial break. The default setting, 2152*, assigns ALT-F1. (Press the F1 function key while holding ALT.)</p> <p>The scan codes used for this setting do <u>not</u> correspond to keyboardscan codes documented in IBM personal computer manuals. Use Show Scan Code, described below, to determine the correct setting for the key or key combination (Shift, CTRL or ALT plus another key) you select.</p> <p>PCM MegaBasic does <u>not</u> respond to Break, although user programs can.</p> |
| Break Delay | This delay value is arbitrary and varies among different computer models. It is a count of the number of times a software delay loop is repeated between the start and end of a break. The default is 20,000* counts. |
| Show Scan Code | This menu item is shown after you answer Y to the prompt that asks whether you wish to change a selection. You should use Show Scan Code to discover the scan code for the key or key combination you want to assign to the Send Break function or exit TERMF. |

* Default selection.

Caution

The key assignment for exiting from TERMF should never be changed when using PCOP. A change to this setting will prevent PCOP from functioning as expected.

When these selections are correct, type **N**.

6. Menu item 4 selects a screen that displays the current values for display attributes. These values select the foreground and background colors for characters based on the status of the graphics rendition modes reverse, intensify, underline, and blink. This selection allows you to change the colors of the display used by TERMF and PCOP. The values are coded as two hexadecimal digits representing a string of eight bits. The three least significant bits (1 through 3) code the foreground color; bit 4 specifies high intensity; bits 5 through 7 code the background color; and bit 8 specifies blinking.

TERMSET shows the names of the colors which correspond to the numeric values selected for foreground and background colors. These are the colors displayed on CGA, EGA and VGA color displays. If your computer has a monochrome display, you should use the default display attributes.

To redefine any of the display attributes, type **Y** and respond to the prompts for attribute combination number and hexadecimal attribute value. When the settings are correctly displayed, type **N**.

7. Menu items 5, 6, and 7 select screens that display the scan codes contained in the normal, application, and cursor key binding sets, respectively. Key bindings are used to redefine keys for various operational modes. A single keystroke may send up to 80 characters via these key bindings.

Note

Custom key bindings have limited value when using TERMF or PCOP with the PCM. This description is provided only for completeness.

When using MegaBasic special key handling, only ordinary keys (i.e., those that produce a non-null character code) can be used for keybindings. For example, CTRL-A can be used to produce the character code 1.

If MegaBasic special key handling is selected from main menu item 3, many of these custom binding in these sets will be replaced with MegaBasic defaults.

To view or modify the normal key bindings, type **Y**, select the appropriate function, and answer the prompts:

| | |
|---|---------------------------------|
| 1 | Look at a table entry. |
| 2 | Replace a table entry. |
| 3 | Delete a table entry. |
| 4 | Add a table entry. |
| 5 | Show the scan code for the key. |

When the key bindings are correct, type **N**.

8. Menu item 8 selects a screen which displays the current values for the display color palette settings. These settings are only available for EGA/VGA adapters. The EGA color palette has 16 entries. The colors are defined in RGB notation (i.e., three digits with values 0 to 3 each to indicate the intensity of Red (first digit), Green (second digit), and Blue (third digit). The range for each color is 000 to 333, giving a palette of 64 possible colors.

If you wish to change any of these settings, type **Y** and respond to the prompts for register number, RGB value, and color name. If the settings are correct as displayed, type **N**.

Local Configuration File

If you frequently use more than one TERMF or PCOP setup, you can save each configuration in its own file. You can specify the local configuration file by naming it when you invoke **TERMSET**, as follows:

```
TERMSET [new filename]
```

For example, to switch between using an EGA and monochrome display, type:

```
COPY DEFAULT.DAT TERM.EGA
TERMSET TERM.EGA
```

and modify the default settings to use your EGA display. When you exit from **TERMSET**, these settings are saved to the file **TERM.EGA**. Then, create a file named **TERM.MON** to describe the monochrome display by typing:

```
COPY DEFAULT.DAT TERM.MON
TERMSET TERM.MON
```

Whenever you need to change your configuration to the EGA setup, type:

```
COPY TERM.EGA TERM.DAT      or      PCOP
TERMF
```

Or, for the monochrome monitor, type:

```
COPY TERM.MON TERM.DAT      or      PCOP
TERMF
```

You could also specify which file to use when you invoke **TERMF** or **PCOP**, as follows:

```
TERMF TERM.EGA      or      PCOP TERM.EGA      or
TERMF TERM.MON      or      PCOP TERM.MON
```

Connecting the PCM to the Programmer

Cables for connecting the PCM to a display terminal or personal computer are described in appendix A, *PCM Cabling Information*. To connect the PCM to the programmer:

1. Connect the cable between the PCM programmer port (usually port 1) and the serial port on the VT100™ or OIT terminal or IBM PC-XT, PC-AT, PS/2, Workmaster, Workmaster II, or Cimstar I industrial computer.

If you are using a computer as the programmer, be sure the cable is connected to the serial port (COM1 or COM2) specified by **TERM.DAT**. If you are using PCOP, refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487, for additional information on establishing communication with the PCM.

2. If you are using a computer as the programmer, type **TERMF** at the MS-DOS prompt and press the Enter key.
3. Press the PCM Restart/Reset pushbutton for 10 seconds to initiate a hard reset and place the PCM in programming (factory) mode.
4. If the PCM has firmware version 2.50 or greater and has been configured using Logicmaster 90 software for BASIC or BAS/CCM mode, the MegaBasic banner should be displayed at the top of the screen, followed by the "Ready" prompt.

```
MegaBasic Version 5.602, under PCM VTOS v2.50
IEEE/Software floating point on an 80186/88 CPU

(c)Copyright 1985-1990 by Christopher Cochran
MegaBasic Support BBS: 415-459-0896, PO Box 723
Fairfax, CA. USA 94930 - Serial #0000

Ready
```

With other Logicmaster 90 configuration modes, you should see a "->" prompt from the PCM command interpreter. If your PCM has firmware version 2.04 or lower, you must use PCOP to program the PCM.

™ VT100 is a trademark of Digital Equipment Corporation.

Diagnosing Serial Communication Problems

Before proceeding, verify that the OK LED on the PCM is on. If the LED is off, refer to the information in section 1 of this chapter.

This procedure is used to determine if there is a hardware problem with the programmer serial ports.

1. Verify that both the PCM and the programmer are using the same baud rate, parity, number of data bits, number of stop bits, and the same type of handshaking (**HARDWARE**, **SOFTWARE**, or **NONE**).

Note

If you configured the PCM using Logicmaster 90 software, review section 2 of this chapter. If you used the Series 90-30 Hand-Held Programmer, review section 3 of this chapter. When verifying the programmer serial port configuration, review section 5 of this chapter.

2. Verify that the cable connections, described in appendix A, *PCM Cabling Information*, are correct and that the cable is firmly secured at both ends.
3. Press the PCM Restart/Reset pushbutton for 10 seconds. The middle light on the PCM should blink. If it does not, remove the connector from the PCM, jumper pins 4 and 5 on the PCM with a paper clip, and press the Reset/Restart pushbutton again for 10 seconds. If the LED still does not blink at least once, there is a problem with the PCM. Otherwise, the cable, programmer configuration, or programmer hardware is the problem; continue with step 4.
4. Reconnect the cable to the PCM. If the programmer has more than one serial port, be sure the cable is connected to COM1. Set the programmer serial port to the PCM default settings. To do this when using a computer as the programmer, type **TERMF** **DEFAULT.DAT** at the MS-DOS prompt and press the Enter key.
5. Press and hold the Restart/Reset pushbutton for 10 seconds to initialize the PCM to its factory default settings.
6. Press the programmer Enter key while watching the USER1 LED for serial port 1 or USER2 LED for serial port 2. Each time the key is pressed the LED should blink. If the PCM has been configured by Logicmaster 90 in BASIC or BAS/CCM mode, the "Ready" prompt should also be repeated on the programmer screen; otherwise the ">" prompt should appear. If the LED does not blink or the "Ready" or ">" prompt is not displayed, either the connection from the programmer to the PCM is bad or the programmer hardware is defective.
7. Cycle power on the programmer to make sure its serial port hardware is fully reset. Problems with the programmer are very rare. When they do occur, they can often be fixed with a power cycle. If your programmer is a computer, type **TERMF** **DEFAULT.DAT** again. If the LED still does not blink when a key is pressed, it is likely that there is a problem with the cable. See appendix A, *PCM Cabling Information*, for information on PCM cables. Occasionally, problems occur with the PCM or PC serial port hardware.
8. Press CTRL-BREAK or ALT-Z to exit TERMF.

If PCOP establishes communication with the PCM but the PCOP status never switches to **ONLINE**, refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487, for additional suggestions.

Chapter 3

CCM Operation

This chapter contains information relevant to the operation of the CCM communication protocol on the Series 90 PCM. The PCM must be configured for CCM operation, using Logicmaster 90 configuration software or PCOP, before attempting CCM communication. Refer to chapter 2, section 2 of this manual for a guide to configuring the PCM.

This chapter contains the following sections:

| Section | Title | Description | Page |
|---------|---|--|------|
| 1 | Series 90 CCM Target Memory Types | Section 1 defines the memory types for the Series 90 PLC. Memory allocation for the CCM scratch pad and diagnostic status words is also described in this section. | 3-2 |
| 2 | Series 90 CCM Memory Addressing Conventions | Section 2 explains the addressing conventions and data lengths for each memory type. | 3-7 |
| 3 | Communications Request (COMMREQ) | Section 3 describes the Communications Request (COMMREQ) function. | 3-12 |
| 4 | CCM COMMREQ Data Block | Section 4 describes the CCM COMMREQ data block and includes a summary of the data blocks for the Series 90 command set. | 3-19 |
| 5 | CCM COMMREQ Status Word | Section 5 describes the status word returned by a CCM COMMREQ. Also included is a table of return status error codes. | 3-23 |
| 6 | CCM COMMREQ Example | Section 6 provides a complete Series 90 PLC ladder logic program containing CCM COMMREQ function blocks for the PCM. The sample program is suitable for both Series 90-30 and Series 90-70 PLCs. | 3-25 |
| 7 | PLC System Communications Window | Section 7 describes the effect this configurable portion of the Series 90-70 PLC CPU execution sweep can have on CCM communication. Also included is how to adjust the window time from the PLC program. | 3-28 |

Comparisons of the CCM implementations in Series 90™, Series Six™, Series Five™, and Series One™ PLCs are included in sections 1 and 2 of this chapter to assist those who are already experienced in the operation of CCM on other GE Fanuc programmable logic controllers. If you need information on Series 90 CCM only, skip to section 3 of this chapter.

Section 1: Series 90 CCM Target Memory Types

Series 90 CCM supports a subset of the memory types available in Series Six CCM. Tables 3-1 and 3-2 below list the Series 90 CCM memory types. The types for the CCM single bit write function (listed in table 3-2) are functional memory types. They map to the same input and output tables as memory types 1 and 2, but are assigned unique memory type numbers because they are used to perform bit set and bit clear special operations on the input and output tables.

Table 3-1. Memory Types Supported by Series 90 CCM

| CCM Memory Type | CCM Target Table |
|-----------------|-------------------------|
| 1 | Register Table (%R) |
| 2 | Input Table (%I) |
| 3 | Output Table (%Q) |
| 6 | CCM Scratch Pad |
| 9 | Diagnostic Status Words |

Table 3-2. Memory Types for the CCM Single Bit Write Function (6110)

| CCM Memory Type | CCM Target Table | Bit Operation |
|-----------------|-------------------|---------------|
| 13 | Input Table (%I) | Bit Set |
| 14 | Output Table (%Q) | Bit Set |
| 17 | Input Table (%I) | Bit Clear |
| 18 | Output Table (%Q) | Bit Clear |

Those Series Six memory types that are not supported are listed in the following table.

Table 3-3. Series Six Memory Types NOT Supported by Series 90 CCM

| CCM Memory Type | CCM Target Table | Bit Operation |
|-----------------|-----------------------|---------------|
| 0 | Absolute | – |
| 4 | Input Override Table | – |
| 5 | Output Override Table | – |
| 7 | User Logic | – |
| 8 | Quick Access Buffer | – |
| 10 | Timers | – |
| 11 | Counters | – |
| 15 | Input Override Table | Bit Set |
| 16 | Output Override Table | Bit Set |
| 19 | Input Override Table | Bit Clear |
| 20 | Output Override Table | Bit Clear |
| 21 | Input Override Table | Bit Toggle |
| 22 | Output Override Table | Bit Toggle |

The next two tables compare the Series One and Series Five CCM memory types with those supported by the Series 90 CCM.

Table 3-4. Series One Memory Types vs. Series 90 CCM Memory Types

| Series One CCM | | Series 90 CCM | |
|----------------|--------------------------------------|---------------|---------------------------|
| Memory Type | Target Table | Memory Type | Target Table |
| 1 | Timer/Counter/Data Register | 1 | Register Table |
| 3 | Discrete I/O ¹ | 2, 3 | Input Table, Output Table |
| 6 | Scratch Pad ² | 6 | CCM Scratch Pad |
| 7 | User Logic | Not Supported | |
| 9 | Diagnostic Status Words ³ | 9 | Diagnostic Status Words |

¹ The addressing scheme for the Series One PLC differs from that of the Series 90 PLC when accessing I/O points. Refer to the next section for more information.

² Scratch pad definitions are not the same in the Series One PLC and the Series 90 PLC. See table 3-6 for the Series 90 scratch pad layout.

³ Diagnostic status words and error code definitions are different in the Series One PLC and the Series 90 PLC. See table 3-7 for the Series 90 diagnostic status words, and refer to table 3-12 for the Series 90 CCM error code definitions.

Table 3-5. Series Five Memory Types vs. Series 90 CCM Memory Types

| Series Five CCM | | Series 90 CCM | |
|-----------------|--------------------------------------|---------------|-------------------------|
| Memory Type | Target Table | Memory Type | Target Table |
| 1 | Registers | 1 | Register Table |
| 2 | Inputs ^{1,3} | 2 | Input Table |
| 3 | Outputs ^{2,3} | 3 | Output Table |
| 6 | Scratch Pad ⁴ | 6 | CCM Scratch Pad |
| 7 | User Logic | Not Supported | |
| 9 | Diagnostic Status Words ⁵ | 9 | Diagnostic Status Words |

¹ The Series Five local and special inputs do not exist in the Series 90 input table. All inputs are equivalent, and it is up to you to determine their functionality.

² The Series Five local and internal outputs do not exist in the Series 90 output table. All outputs are equivalent, and it is up to you to determine their functionality. For example, an output to be used as an internal coil must not be tied to a real output.

³ The addressing scheme for the Series Five PLC differs from that of the Series 90 PLC when accessing I/O points. Refer to the next section for more information.

⁴ Scratch pad definitions are not the same in the Series Five PLC and Series 90 PLC. See table 3-6 for the Series 90 scratch pad layout.

⁵ Diagnostic status words and error code definitions are different in the Series Five PLC and the Series 90 PLC. See table 3-7 for the Series 90 diagnostic status words, and refer to table 3-12 for the Series 90 CCM error code definitions.

CCM Scratch Pad

The entire scratch pad is updated every time an external READ request is received by CCM with a memory type of 6. All scratch pad locations are **read only**. The scratch pad is a byte-oriented memory type.

Table 3-6. Scratch Pad Memory Allocation

| SP Address | Field Identifier | Bits | | | | | | | |
|------------------|---------------------------|---|---|---|---|--------------|---|---|---|
| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 00 | CPU Run Status | 0 | 0 | 0 | 0 | See Note (1) | | | |
| 01 | CPU Command Status | Bit pattern same as SP(00) | | | | | | | |
| 02 | CPU Type | Major ^{2a} (in hexadecimal) | | | | | | | |
| 03 | | Minor ^{2b} (in hexadecimal) | | | | | | | |
| 04 – 0B | CPU ID | 7 ASCII characters + termination character = 0 | | | | | | | |
| 0C | CPU Firmware Revision No. | Major (in BCD) | | | | | | | |
| 0D | | Minor (in BCD) | | | | | | | |
| 0E | PCM Firmware Revision No. | Major | | | | | | | |
| 0F | | Minor | | | | | | | |
| 10 – 11 | Reserved | (00H) | | | | | | | |
| 12 ^{3a} | Node Type Identifier | (90-70: 0CH; 90-30: 0DH) ^{3b} | | | | | | | |
| 13 – 15 | Reserved | (00H) | | | | | | | |
| 16 | CCM CPU ID | Master/Slave: 1 – 90 (decimal) Peer-to-Peer: 1 – 254 Universal Responder: 255 | | | | | | | |
| 17 | Reserved | (00H) | | | | | | | |
| 18 – 33 | Sizes of Memory Types | See Note (4) | | | | | | | |
| 18 – 1B | Register Memory | %R size | | | | | | | |
| 1C – 1F | Analog Input Table | %AI size | | | | | | | |
| 20 – 23 | Analog Output Table | %AQ size | | | | | | | |
| 24 – 27 | Input Table | %I size | | | | | | | |
| 28 – 2B | Output Table | %Q size | | | | | | | |
| 2C – 2F | Internal Discrete Memory | %M size | | | | | | | |
| 30 – 33 | User Program Code | See Note (5) | | | | | | | |
| 34 – FF | Reserved | (00H) | | | | | | | |

Scratch Pad Memory Allocation Footnotes (for table 3-6)

- ¹ 0000 = Run_Enabled 0100 = Halted
 0001 = Run_Disabled 0101 = Suspended
 0010 = Stopped 0110 = Stopped_IO_Enabled
 0011 = Stopped_Faulted
- ^{2a} PLC CPU Major Type Codes:
 S9070_PLC_CPU 12 (0ch) Series 90-70 PLC CPU
 S9030_PLC_CPU 16 (10h) Series 90-30 PLC CPU
- ^{2b} Series 90-70 Minor Types for CPU:
- | | |
|-------------------------------------|--|
| CPU_731 31 (1Fh) Series 90-731 CPU. | CPU_790 90 (5ah) Series 90-790 CPM CPU |
| CPU_732 32 (20h) Series 90-732 CPU | CPU_915 15 (0fh) Series 90-915 CPM CPU |
| CPU_771 71 (47h) Series 90-771 CPU | CPU_925 25 (19h) Series 90-925 CPM CPU |
| CPU_772 72 (48h) Series 90-772 CPU | CPX_772 73 (49h) Series 90-772 CPX CPU |
| CPU_780 80 (50h) Series 90-780 CPU | CPX_782 83 (53h) Series 90-782 CPX CPU |
| CPU_781 81 (51h) Series 90-781 CPU | CPX_928 28 (1ch) Series 90-928 CPX CPU |
| CPU_782 82 (52h) Series 90-782 CPU | CPX_935 35 (23h) Series 90-935 CPX CPU |
| CPU_788 88 (58h) Series 90-788 CPU | CGR_772 74 (4ah) Series 90-772 CGR CPU |
| CPU_789 89 (59h) Series 90-789 CPU | CGR_935 36 (24h) Series 90-935 CGR CPU |
| CPU_914 92 (5ch) Series 90-914 CPU | |
| CPU_924 24 (18h) Series 90-924 CPU | |
- Series 90-30 Minor Types for CPU:
- | | |
|--|---------------------------------------|
| CPU_311 30 (1eh) Series 90-30 311 CPU | CPU_350 44 (2ch) Series 90-30 350 CPU |
| CPU_313 33 (21h) Series 90-30 313 CPU | CPU_351 37 (25h) Series 90-30 351 CPU |
| CPU_323 34 (22h) Series 90-30 323 CPU | CPU_352 39 (27h) Series 90-30 352 CPU |
| CPU_331 35 (23h) Series 90-30 331 CPU | CPU_360 40 (28h) Series 90-30 360 CPU |
| CPU_340 38 (26h) Series 90-30 340 CPU | CPU_363 41 (29h) Series 90-30 363 CPU |
| CPU_341 36 (24h) Series 90-30 341 CPU. | CPU_364 42 (2ah) Series 90-30 364 CPU |
- ³ Located in the same position as in the Series Six scratch pad. Series One, Three and Five PLC users, who need to determine the node type, should note this location and make driver modifications where necessary.
- ⁴ Scratch Pad Bytes 18h–33h:

| Bytes | Length of Memory | Size Returned In |
|-------|-----------------------------|------------------|
| 18–1B | %R Register Memory | Words |
| 1C–1F | %AI Analog Input Table | Words |
| 20–23 | %AQ Analog Output Table | Words |
| 24–27 | %I Input Table | Points (Bits) |
| 28–2B | %Q Output Table | Points (Bits) |
| 2C–2F | %M Internal Discrete Memory | Points (Bits) |
| 30–33 | User Program Code | Bytes |

Note: Four bytes hold the hexadecimal length of each memory type with the most significant word reserved for future expansion. For example, the 731 default register memory size of 1024 words (0400h) would be returned in the following format:

| Word | Least Significant | | Most Significant | |
|----------|-------------------|----|------------------|----|
| SP Byte | 18 | 19 | 1A | 1B |
| contains | 00 | 04 | 00 | 00 |

- ⁵ The amount of program memory occupied by the logic program. Also appears on the Logicmaster 90 PLC Memory Usage screen in the **User Program** field.

Diagnostic Status Words

In addition to the status word, which is automatically transferred from the CCM task to the CPU, there are 20 diagnostic status words maintained and updated within CCM. These status words are not automatically transferred to the CPU; the internal COMMREQ command 6003 (Read Diagnostic Status Words to Source Registers) is used to transfer these status words to the CPU. An external device can access these status words using a READ command with target memory type 9. Table 3-7 explains the purpose of each diagnostic status word.

When CCM runs concurrently on both PCM serial ports, each has its own copy of diagnostic status words. Neither can report on the status of the other.

The Series Six diagnostic status words contain data referring to both ports. Because the Series 90 maintains two separate sets of diagnostic status words, the words referring to the other port have been removed in the Series 90 and the diagnostic status words reorganized, as outlined in the following table. The serial configuration data for both ports has also been removed. The software version number remains in the same location as in the Series Six PLC.

Table 3-7. CCM Diagnostic Status Word Definitions

| Diagnostic Status Word | Word Contents | |
|------------------------|---|-------------------------------------|
| | Byte 2 | Byte 1 (LSB) |
| 1 | 00H | Serial Port Error Code ¹ |
| 2 | Number of Successful Conversations ² | |
| 3 | Number of Aborted Conversations ² | |
| 4 | Number of Header Retries | |
| 5 | Number of Data Block Retries | |
| 6 | Number of Q-Sequence Successes | |
| 7 | Number of Peer-to-Peer Collisions | |
| 8 – 11 | Reserved (00H) | |
| 12 | CCM Software Version Number ³ | |
| 13 | COMMREQ Error Code ⁴ | |
| 14 | Reserved (00H) | |
| 15 – 20 | COMMREQ Data Block Contents | |

¹ See table 3-12, CCM Serial Port Error Codes, for a list of the possible error codes and their definitions.

² Internal commands do not modify this count. The term “conversation” refers to communications across the serial port.

³ Same as the PCM Firmware Revision Number in the scratch pad (0E-0F). This value always remains in word 12 of the diagnostic status words, even when the diagnostic status words are cleared by issuing internal command 6002 or by an external device request.

⁴ Refer to section 5 of this chapter for a description of the returned status for a CCM COMMREQ.

Section 2: Series 90 CCM Memory Addressing Conventions

In order to carry out a data transfer, the CCM protocol must be given the address where the transfer is to begin and the length of the data to be transferred. The starting address plus the length must not go past the end of a table boundary. The requirements for specification of the starting address and data length are explained in this section, followed by general guidelines for replacing a Series One, Series Three, Series Five or Series Six PLC with a Series 90 PLC in an application using the CCM protocol.

Target/Source Memory Addresses

The memory addresses in the following table are target addresses when the non-initiating device is a Series 90 PLC. When the initiating device is a Series 90 PLC, they are source addresses.

Table 3-8. Target/Source Memory Addresses

| Memory Type | Description | Address Ranges ¹ |
|---|---|-----------------------------|
| 1 Register | Specified the register with which the data transfer is to begin. | 1–Maximum Units |
| 2 Input Table 3 Output Table | Specifies the input or output point with which the data transfer is to begin. Source memory address must be on a byte boundary (i.e., 1, 9, 17 ...). ² | 1–Maximum Units |
| 6 CCM Scratch Pad Memory ³ | Specifies the scratch pad byte with which the data transfer is to begin. | 0–255 |
| 9 CCMDiagnostic Status Words ³ | Specifies the diagnostic status word with which the data transfer is to begin. | 1–20 |
| 13 Bit Set Input 14 Bit Set Output | Specifies the input or output point to be set. | 1–Maximum Units |
| 17 Bit Clear Input 18 Bit Clear Output | Specifies the input or output point to be cleared. | 1–Maximum Units |

¹ The maximum addressable ranges for each memory type depends on the model of CPU and memory configuration.

² For I/O references, the Series 90 and Series Six CCM implementations use point-oriented addressing, rather than the byte-oriented addressing of the Series One, Three and Five PLCs. The starting address is interpreted by the Series 90 PLC as the bit number at which the transfer is to begin. Series 90 source memory addresses must be on a byte boundary. (See the examples that follow.)
Software packages which use the byte-oriented addressing method to interface with a Series One, Three, or Five PLC may need to be modified for the Series 90 PLC.

³ Scratch pad and diagnostic status words are resident in PCM/CMM memory.

Examples:

To read target Series 90 inputs 9 through 16 into source Series 90 inputs 17 through 24, the source address is 17, the target address is 9, and the data length is 8.

To read target Series One inputs 9 through 16 into source Series 90 inputs 17 through 24, the source address is 17, the target address is 2 (Series One I/O addressing is byte-oriented), and the data length is 8.

To read target Series 90 input 27 into source Series 90 input 3, you must specify a source address of 1, a target address of 25, and a data length of 8. Inputs 1 through 8 of the source input table are overwritten with the values of inputs 25 through 32 of the target input table.

To read target Series One input 27 into source Series 90 input 3, you must specify a source address of 1, a target address of 4, and a data length of 8. Inputs 1 through 8 of the source input table are overwritten with the values of inputs 25 through 32 of the target input table.

Data Length

Data length refers to the length of the data transfer. The units are determined by the source memory type and are listed in the following table.

Table 3-9. Unit Lengths of Series 90 CCM Memory Types

| Memory Type | Unit Length | Length Accessible |
|--|----------------------|-------------------------|
| 1: Registers | 1 Register = 16 bits | Register(s) |
| 2,3: Inputs and Outputs | 1 Point = 1 bit | Multiple(s) of 8 Points |
| 6: Scratch Pad | 1 Byte = 8 bits | Byte(s) |
| 9: Diagnostic Status Words | 1 Word = 16 bits | Word(s) |
| 13,14: BitSetInputs/Outputs 17,18: BitClearInputs/Outputs | 1 Point = 1 bit | 1 Point |

Examples:

To read 12 bytes of the target Series 90 scratch pad into Series 90 (or Series Six) registers, the data length is 6 since the unit length for the source memory type (registers) is a register. To read 12 diagnostic status words into the registers, the data length would be 12 because both registers and diagnostic status words have equivalent unit lengths (register = word = 2 bytes).

To read 8 target Series 90 inputs into Series 90 (or Series Six) inputs, the data length is 8 points since the unit length is the same for each. CCM memory types 2 and 3 (inputs and outputs) can be accessed only in multiples of 8.

To read 8 target Series 90 registers into Series 90 (or Series Six) inputs, the data length is 8 registers times 16 points per register = 128 points.

CCM Comparisons

The following diagrams compare the CCM implementations on the Series 90 PLC with those on the Series One™, Series Three™, and Series Five™ PLCs.

The mapping of the Series 90 is the same as that of the Series Six™ PLC. Note, however, that memory organization within the diagnostic status words and scratch pad differs between Series 90 PLCs and Series Six PLCs.

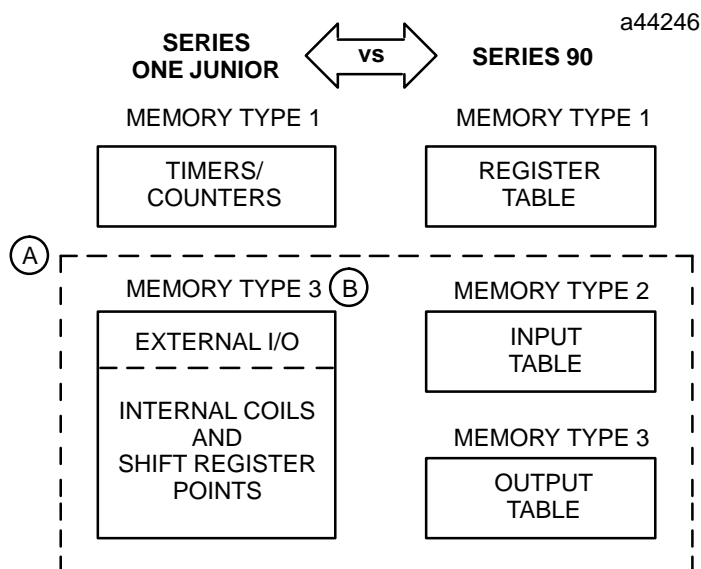


Figure 3-1. Series One Jr. PLC vs Series 90 PLC

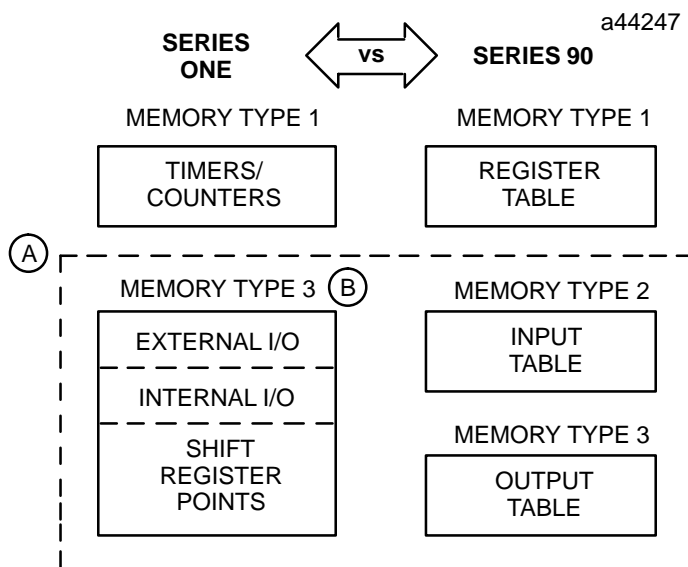


Figure 3-2. Series One PLC vs Series 90 PLC

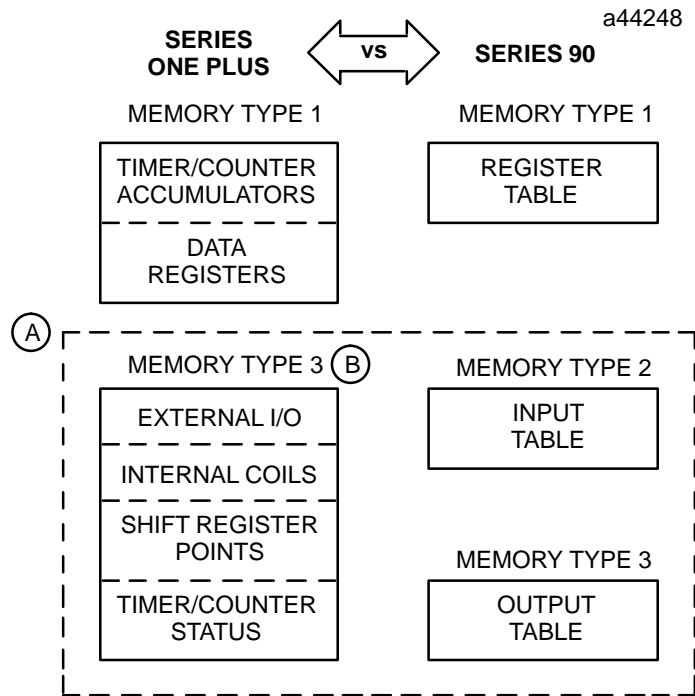


Figure 3-3. Series One Plus PLC vs Series 90 PLC

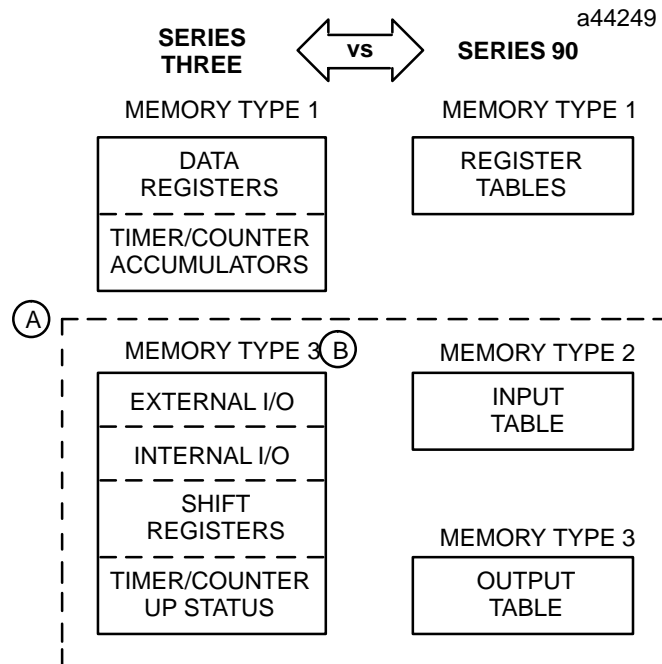


Figure 3-4. Series Three PLC vs Series 90 PLC

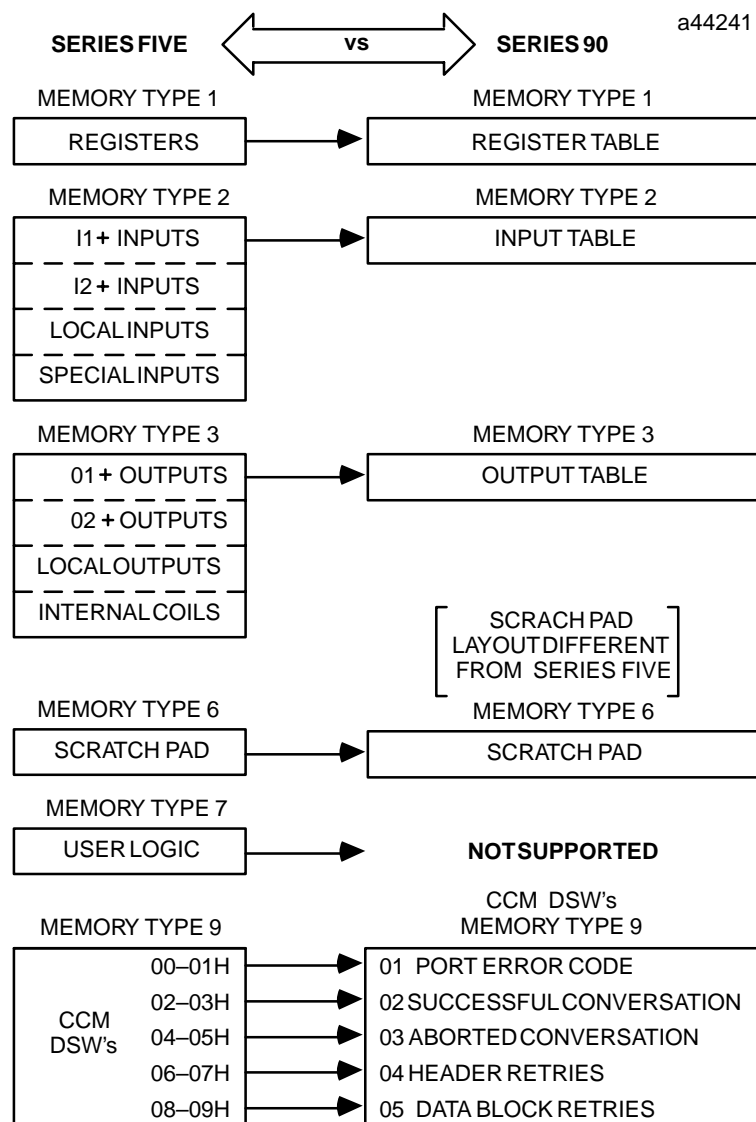


Figure 3-5. Series Five PLC vs Series 90 PLC

Notes Relating to Figures 3-1 through 3-5:

- A. The Series One and Series Three PLCs have one table for I/O. The Series 90 PLC has two separate tables, one for inputs and one for outputs. Depending upon the I/O type of the data being retrieved, it is necessary to use memory type 2 or 3, instead of just 3.
- B. The Series One, Series Three, and Series Five PLCs use byte-oriented addressing (I17: start address = 3). The Series Six and Series 90 use bit-oriented addressing (I17: start address = 17).

Section 3: Communications Request (COMMREQ)

When a PCM is configured as a CCM master or peer, it can initiate CCM messages only when the PLC CPU commands it to do so. Communications Request (COMMREQ) function blocks in the ladder logic program are used for this purpose. The PLC CPU uses the parameters of a COMMREQ and its associated command block to send a command to the PCM. The PCM, in turn, sends the CCM message that was specified in the command.

When a COMMREQ function block receives power flow, the CPU may send a CCM command and wait for a reply. This mode of operation is referred to as **WAIT mode**. The maximum length of time the PLC will wait for CCM to respond is specified in the command block of the COMMREQ. If CCM does not respond within that time, program execution resumes.

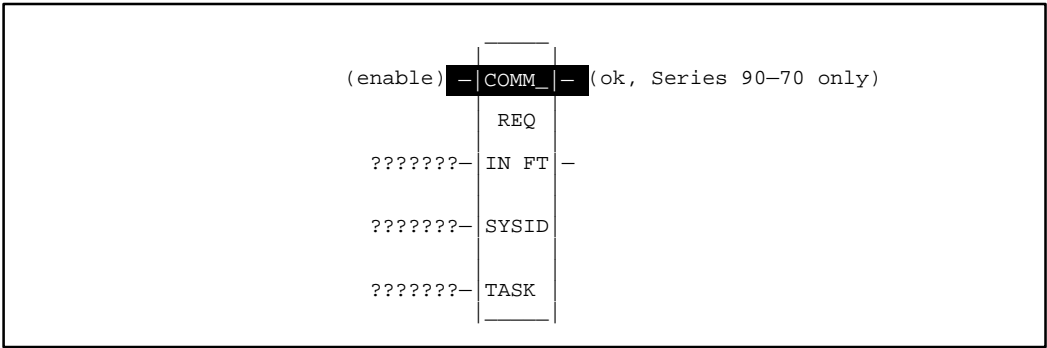
If the COMMREQ command block specifies that the PLC program should not wait for a reply, a CCM command is sent and program execution resumes immediately. The timeout values for the command block are ignored. This mode is referred to as **NOWAIT mode**.

Caution

NOWAIT mode should always be used. Otherwise, the time spent waiting for CCM communication will significantly degrade PLC sweep time. If WAIT mode is used, the sum of the maximum PLC sweep time plus the longer of the two COMMREQ timeout values must be no larger than the PLC watchdog timer setting. Information on the two COMMREQ timeout values is presented later in this section.

Format of the COMMREQ Function Block

The COMMREQ function block has four inputs which specify the command block location in PLC memory and the identity of CCM. A fault output indicates errors. The Series 90-70 COMMREQ function block also has an OK output; the Series 90-30 COMMREQ does not.



- IN:** Specifies the memory location of the command block. It may be any word-oriented user reference (%R, %AI, or %AQ in both the Series 90-70 and Series 90-30 COMMREQ; %P or %L in the Series 90-70 COMMREQ only).
- SYSID:** SYSID is a hexadecimal value containing the rack and slot location of the PCM to which the COMMREQ is being sent. Entries have this format:



If SYSID is incorrectly programmed for a rack and slot which does not contain a PCM or other intelligent module, no communications request will be sent, the OK output (if any) will remain inactive, and the FT output will become active.

Note

No error occurs if SYSID specifies a rack and slot which contains an intelligent module that is not a PCM.

Additional examples:

| Rack | Slot | Hexadecimal Word Value |
|------|------|------------------------|
| 0 | 4 | 0004h |
| 3 | 4 | 0304h |
| 2 | 9 | 0209h |
| 7 | 2 | 0702h |

TASK: The following table lists the applicable task numbers for CCM.

| Task Number | Description |
|-------------|-------------------|
| 1 | CCM on PCM Port 1 |
| 2 | CCM on PCM Port 2 |

If the task number programmed for CCM is not valid, a COMMREQ BAD TASK ID application fault is logged in the PLC fault table. This can occur if the TASK value is misprogrammed, if the PCM has been hard reset so that CCM is inactive, or if the PCM is not configured correctly for the specified CCM task.

OK and FT: The OK and FT (function faulted) outputs can provide power flow to optional logic to verify successful completion of the COMMREQ. Note that the Series 90-30 COMMREQ has no OK output. OK and FT may have these states:

| ENable | Error? | OK output | FT output |
|------------|--------------|-----------|-----------|
| active | no | true | false |
| active | yes | false | true |
| not active | no execution | false | false |

In **NOWAIT** mode, a COMMREQ always passes power flow to the OK output whenever it executes. In **WAIT** mode, the function passes power flow to the OK output unless the timeout expires before the COMMREQ is completed or a zero timeout period is specified. Then, OK remains inactive and FT becomes active.

The FT output also becomes active in **WAIT** or **NOWAIT** mode if:

- There is no PCM or other intelligent module in the rack and slot specified by the SYSID input.
- The data length specified in the command block is zero.

In **WAIT** mode, the FT output also becomes active if:

- The PCM port specified by the TASK input is not configured as a CCM master or peer, CCM is not enabled, or the PCM has been hard reset. In either **WAIT** or **NOWAIT** mode a BAD TASK ID fault is logged in the PLC fault table.

If there are errors in the portion of the command block used specifically by CCM, these errors are reflected in the value returned in the status location, and not in the FT output.

Other COMMREQ Faults

If the CCM status pointer address specified in the command block does not exist, the status information returned by CCM will be lost. This may occur when either the user reference type specified in the command block or the offset address within that reference type is invalid.

If CCM receives COMMREQs from the PLC faster than they can be processed, a MAILBOX QUEUE FULL fault may eventually be logged in the PLC fault table:

MOD: Other S/W error COMMREQ MB FULL START

It is good programming practice to have no more than one COMMREQ outstanding at a time on each CCM port. The example CCM application in section 6 of this chapter shows how the CCM status register can be used to send one COMMREQ at a time.

MAILBOX QUEUE FULL faults can also occur if CCM has been stopped by a hard reset or if the PCM has stopped functioning.

Power-Up Delay

The first COMMREQ sent to CCM after a power cycle or ACFAIL must be delayed until the PCM has finished power-up initialization. A good rule of thumb is to wait 5 seconds before trying to send a CCM COMMREQ. The absolute minimum time is 1 second. Refer to the example in section 6 of this chapter.

Command Block

The command block provides additional information needed by the COMMREQ function.

The address of the command block is specified for the IN input to the COMMREQ function. This address may be in any word-oriented user reference (%R, %AI or %AQ in both the Series 90-70 and Series 90-30 COMMREQ; %P or %L in the Series 90-70 COMMREQ only). The length of the command block depends on the specific CCM command being sent.

The command block has the following structure:

| | |
|----------------------------|-----------------------------------|
| Data Block Length | address (word 1) |
| Wait/NoWait Flag | address + 1 (word 2) |
| Status Pointer Memory Type | address + 2 (word 3) |
| Status Pointer Offset | address + 3 (word 4) |
| Idle Timeout Value | address + 4 (word 5) |
| Maximum Communication Time | address + 5 (word 6) |
| Data Block | address + 6 (word 7) |
| | through address + 11 (word 12) |

Information required for the command block can be placed in the designated memory area using the MOV or BLKMOV function block.

When entering information for the command block, refer to these definitions:

- Data Block Length:** This word contains the number of data words starting with the CCM command number at address + 6 (word 7) to the end of the command block, inclusive. The data block length of CCM commands ranges from 1 to 6 words. Each CCM command has its own COMMREQ data block length.
- Wait/No Wait Flag:** This word selects whether or not the program should wait for CCM communications to be completed.

| For | Enter |
|----------------|-------|
| No wait | 0 |
| Wait for reply | 1 |

The flag bit is stored in the least significant bit (LSB) at address + 1 (word 2). The rest of the word is filled with zeros. If the command block is programmed using integer values, this is taken care of automatically.

Caution

It is recommended that the COMMREQ WAIT/NOWAIT flag be set to NOWAIT. Otherwise, CCM communication time can significantly degrade the PLC sweep time. If WAIT mode is used, the sum of the maximum PLC sweep time plus the longer of the two COMMREQ timeout values must be no larger than the PLC watchdog timer setting. If two or more COMMREQs can occur during a single PLC sweep, the total of the timeout values for all the COMMREQs must be considered. Failure to observe this precaution can cause the PLC watchdog timer to expire, halting PLC execution, when a CCM problem occurs.

Information on the two COMMREQ timeouts is presented below. For information on reading PLC watchdog timer settings and configuring the Series 90-70 watchdog timer, refer to the *Logicmaster 90-70 Programming Software User's Manual*, GFK-0263. The Series 90-30 PLC watchdog timer is not configurable.

**Status
Pointer
Memory
Type:**

The two status pointer words specify a PLC memory location where the status word returned by CCM will be written when the COMMREQ completes.

| | |
|----------------------------|----------------------|
| Status Pointer Memory Type | address + 2 (word 3) |
| Status Pointer Offset | address + 3 (word 4) |

Status pointer memory type contains a numeric code that specifies the user reference memory type for the CCM status word. The table below shows the code for each reference type:

| For This Memory Type | | Use This Value |
|----------------------|-----------------------|----------------|
| %I | Discrete input table | 16 |
| %Q | Discrete output table | 18 |
| %R | Register memory | 8 |
| %AI | Analog input table | 10 |
| %AQ | Analog output table | 12 |

The high byte at address + 2 should contain zero.

**Status
Pointer
Offset:**

The word at address + 3 contains the offset for the CCM status word within the selected memory type.

Note

The status pointer offset is a zero-based value. %R00001, for example, is at offset zero in the register table. %R00300 is at offset 299.

**Idle Timeout
Value:**

The idle timeout value is the maximum time the PLC CPU waits for CCM to acknowledge receipt of the COMMREQ. This value is ignored in **NOWAIT** mode. If **WAIT** mode is selected, address + 4 specifies the idle timeout period in 100-microsecond increments.

**Maximum
Communica-
tion Time:**

The value at address + 5 specifies the maximum time the PLC CPU waits for CCM to complete the COMMREQ. This time is also specified in 100-microsecond increments and is ignored in **NOWAIT** mode.

Data Block:

The CCM data block contains the CCM command number in address + 6 plus any command data words required for each specific command. For more information on CCM COMMREQ data blocks, see section 4 in this chapter.

CCM Status Word

The CCM status word is written to one of five PLC memory types, at the location defined by the status pointer memory type, at address + 2, and the status pointer offset, at address + 3, in the command block. The content of this word is defined as:

| | |
|-----------|--|
| Low byte | Completion code or major error code: 1 indicates success. |
| High byte | Secondary error code. |

Clear the status word before issuing the COMMREQ to CCM. Different memory locations should be used for status words associated with different COMMREQs in order to avoid the possibility of two outstanding COMMREQs writing to the same location. A single status word location should never be used for more than one COMMREQ unless they are all **WAIT** mode requests.

Follow these guidelines for programming the status word:

1. CCM never returns zero for the status word. If the user program needs to know that the command is complete, it can zero the status word before issuing the COMMREQ and then check the status word for a non-zero value.
2. CCM uses a status code value of 1 to indicate that the operation was completed without errors.
3. Display the status word in hexadecimal format to read the two bytes of data. When an error occurs, the least significant byte is greater than 1.

Refer to section 5 of this chapter for a complete list of secondary error codes for CCM.

Section 4: CCM COMMREQ Data Block

Data blocks for CCM on the PCM are similar to the command registers used by the Series Six™ CCM modules. The first word of the data block must be a command word in the range 6000 to 6199 (decimal).

| Subrange | Description |
|-----------|---|
| 6000-6099 | Used for general utility type functions involving only local data storage on the PCM. These commands may be used in all CCM modes (MASTER , SLAVE , PEER-NON-INITIATOR and PEER-INITIATOR). |
| 6100-6199 | Used for operations that require initiating serial communication. These commands are restricted to MASTER and PEER-INITIATOR CCM modes. |

The following table lists the valid command words and the required parameters for each. For more detailed information and examples of each command, refer to the *Series Six™ Data Communications Manual*, GEK-25364.

Table 3-10. COMMREQ Data Block for CCM Commands

| Command Description | Data Block Size | Data Block Words (X indicates “Required”; — indicates “Not Used”) | | | | | |
|---|--------------------------------|--|--|---------------------------|-------------------------------|---------------------|-------------------------------|
| | | Command Word 7 | Target ID Word 8 | Target Memory Type Word 9 | Target Memory Address Word 10 | Data Length Word 11 | Source Memory Address Word 12 |
| Set Q-Response ^{1,2} (SLAVE mode only) | 3 words | 6001 (1770H) | X ^{3a} | X ^{3b} | -- | -- | -- |
| Clear CCM Diagnostic ¹ Status Words | 1 word | 6002 (1771H) | -- | -- | -- | -- | -- |
| Read CCM Diagnostic ¹ Status Words to Source Registers | 6 words (2 words unused) | 6003 (1772H) | | | | | |
| Software Configuration ¹ | 15 words | 6004 (1774H) | See <i>Serial Communication User's Manual</i> (GFK-0582) for details. | | | | |
| Read from Target to Source Register Table | 6 words | 6101 (17D5H) | X | X | X | X | X |
| Read from Target to Source Input Table | 6 words | 6102 (17D6H) | X | X | X | X | X |
| Read from Target to Source Output Table | 6 words | 6103 (17D7H) | X | X | X | X | X |
| Read Q-Response to Source Register Table | 6 words (3 words unused) | 6109 (17DDH) | X | -- | -- | -- | X |
| Single Bit Write | 4 words | 6110 (17DEH) | X | X | X | -- | -- |
| Write to Target from Source Register Table | 6 words | 6111 (17DFH) | X | X | X | X | X |
| Write to Target from Source Input Table | 6 words | 6112 (17E0H) | X | X | X | X | X |
| Write to Target from Source Output Table | 6 words | 6113 (17E1H) | X | X | X | X | X |

¹ Internal Command (no communications across the serial port).

² Only the slave half of the Q-Sequence is supported by Series 90 CCM. The Q-Response may be set via command 6001, and the slave will respond to a Q-Sequence Enquiry received from an external device on the serial port. For a description of the Q-Sequence, refer to pages 4-18 through 4-21 in GEK-25364.

^{3a} Data bytes 1 and 2.

^{3b} Data bytes 3 and 4.

Table 3-11. COMMREQ Data Block for CCM Commands (Explanations for Table 3-10)

| Word | Description |
|-----------------------------------|--|
| Command Word, Word 7 | The COMMREQ data block begins at the seventh word (address + 6) of the command block and consists of up to six contiguous words of memory starting at this location. |
| Target ID, Word 8 | <p>To execute a transfer of data between CCM devices, one CCM device must request the transfer and the other must comply with the request. The device requesting or initiating the transfer is the source; the device complying with, but not initiating, the request is the target. Data may flow from source to target, as well as from target to source.</p> <p>The Target ID is the identification number of the target device; for Series 90 CCM, it is the CCM CPU ID number. Each PCM port can be configured with a different CPU ID number. This number can be assigned using the Logicmaster 90 configuration package. Refer to chapter 2, section 2, <i>Configuring the PCM with Logicmaster 90 Software</i>, for more information on using Logicmaster 90 software to assign the CCM CPU ID.</p> <p>The automatic default configuration provided by the Series 90-30 PLC model 331 CPU sets the CCM CPU ID to 1 on both ports. A CPU ID value of 1 is also the initial default configuration from Logicmaster 90 software.</p> <p>The value of the target ID number may be from 1 to 255 in PEER-TO-PEER mode or from 1 to 90 in MASTER-SLAVE mode. Target ID 0 is reserved. <u>Any peer CCM device, regardless of its ID, responds to target ID 255.</u></p> |
| Target Memory Type, Word 9 | Specifies the type of user reference being accessed in the CCM target device. There are nine accessible Series 90 target memory types (1, 2, 3, 6, 9, 13, 14, 17, and 18). The memory types associated with each number are listed in chapter 3, section 1, <i>Series 90 CCM Target Memory Types</i> . Other CCM devices support different types. |
| Target Memory Address, Word 10 | <p>Specifies the address within the CCM target device where the data transfer is to begin. The address range for each Series 90 memory type is listed in table 3-8.</p> <p><u>Note:</u> For both target memory type and target memory address, error checking is done by the non-initiating device and not by the initiating PCM. Consequently, Series 90 CCM can initiate requests for target memory types and addresses which are invalid for Series 90 targets, as long as the target device is not a Series 90 PLC.</p> |
| Data Length, Word 11 | Specifies the length of the data transfer. The units are determined by the source memory type, which is specified by the command number. Table 3-9 shows the unit length and accessible increment for each memory type. |
| Source Memory Address Word 12 | Specifies the address within the Series 90 CPU where the data transfer is to begin. The source memory address descriptions and ranges are the same as for target memory, as shown in table 3-8. |

The following table summarizes those Series Six commands which are not supported by Series 90 CCM.

Table 3-12. Series Six CCM Commands NOT Supported by Series 90 CCM

| Command Number | Description |
|------------------------------|--|
| 6004 - 6009 6106 6116 | Quick Access Buffer Manipulations |
| 6010 | Set CPU Memory Write Protect |
| 6011 | Reinitialize CCM Task |
| 6012 | Set OIU Timers and Counters |
| 6104 6105 6114 6115 | I/O Override Table Manipulations |
| 6108 6118 6128 | Character String Manipulations (unformatted read/write) |
| 6118 | |
| 6128 | |
| 6117 | Write to Target from Source User Logic |
| 6130 | Set CCM Retries |
| 6131 | Set CCM Timeouts |

Section 5: CCM COMMREQ Status Word

A general description of the status word for a COMMREQ appears in section 3 of this chapter. There are several points to remember when interpreting the contents of the CCM COMMREQ status word:

1. CCM never returns a status word value of zero to the PLC CPU. If the user ladder program needs to know when the command is complete, it can zero the status word before issuing the COMMREQ and then check the status word for a non-zero value.

Note

It is strongly recommended that only one CCM COMMREQ be outstanding for each CCM port at any time. The status word can be used to control the timing of COMMREQs. See the example ladder program in section 6 of this chapter.

2. CCM uses a status code value of 1 to indicate that the operation was completed without errors. Refer to the table below for a complete listing of secondary error codes for CCM.
3. Display the status word in hexadecimal format to read the two bytes of data. When an error occurs, the least significant byte is greater than 1.

The following table lists the CCM serial port error codes that are reported (as secondary error codes) in the most significant byte of the COMMREQ status word after the execution of a CCM COMMREQ. These codes also appear in the least significant byte of CCM Diagnostic Status Word 1.

**Table 3-13. CCM Serial Port Secondary Error Codes
(High Byte of Diagnostic Status Word 1)**

| Error Code | | Description |
|------------|-------------|---|
| Decimal | Hexadecimal | |
| 0 | 00 | Successful transfer. |
| 1 | 01 | A timeout occurred on the serial link. |
| 2 | 02 | A COMMREQ attempted to write data to a section of the CCM scratch pad that is permanently write-protected by the CCM. |
| 3 | 03 | A COMMREQ attempted to read or write a non-existent I/O point. |
| 4 | 04 | A COMMREQ attempted to access more data than is available in a particular memory type. |
| 5 | 05 | A COMMREQ attempted to read or write an odd number of bytes to register memory or the diagnostic status words. |
| 6 | 06 | A COMMREQ attempted to read or write one or more non-existent registers. |
| 7 | 07 | A COMMREQ specified the transfer of zero data bytes. |
| 8 | 08 | A COMMREQ attempted to write to protected memory. |
| 9 | 09 | A COMMREQ attempted to transfer data to or from an invalid memory type or absolute source address. |
| 10 | 0A | A COMMREQ attempted to read or write one or more non-existent diagnostic status words. |

**Table 3-13. CCM Serial Port Secondary Error Codes
(High Byte of Diagnostic Status Word 1) – cont'd**

| Error Code | | Description |
|-------------------|--------------------|--|
| Decimal | Hexadecimal | |
| 11 | 0B | A COMMREQ attempted to transfer data beginning at an invalid scratch pad address or an input/output table address not on a byte boundary (e.g., 1, 9, 17 ...). |
| 12 | 0C | Serial communication was aborted after a data block transfer was retried three times, or a number specified by the configuration. |
| 13 | 0D | Serial communication was aborted after a header transfer was retried three times, or a number specified by the configuration. |
| 14 | 0E | Serial communication was aborted after a Q-Request was retried three times, or a number specified by the configuration. |
| 15 | 0F | An attempt was made to set the Q-Response data on a device not configured as a slave. |
| 20 | 14 | One or more of the following errors occurred during a data block transfer: <ul style="list-style-type: none"> • An invalid STX character was received. • An invalid ETB character was received. • An invalid ETX character was received. • An invalid LRC character was received. • A parity, framing, or overrun error occurred. |
| 21 | 15 | CCM expected to receive an EOT character from an external device and did not receive it. |
| 22 | 16 | CCM expected to receive an ACK or NAK character and did not receive either one. |
| 23 | 17 | Communication was aborted when CCM did not receive a valid acknowledge to a master enquire sequence after 32 attempts, or a number specified by the configuration. |
| 24 | 18 | Communication was aborted after a peer enquire was NAKed 32 times by the external device, or a number specified by the configuration. |
| 25 | 19 | Communication was aborted when the CCM did not receive a valid response to a peer enquire after 32 attempts, or a number specified by the configuration. |
| 26 | 1A | A timeout occurred during an attempt to transmit on a port due to CTS being in an inactive state too long. |
| 29 | 1D | An error occurred when data was being transferred between the CCM and the Series 90 CPU. |
| 30 | 1E | A parity, framing, or overrun error occurred during a serial header transfer. |
| 31 | 1F | A parity, framing, or overrun error occurred during a serial data block transfer. |
| 34 | 22 | Bad Q-Response received. |
| 48 | 30 | COMMREQ attempted to initiate conversation on a port in use. |
| 65 | 41 | The COMMREQ command number is invalid. |
| 66 | 42 | An invalid COMMREQ data block length was specified. |
| 68 | 44 | The COMMREQ is invalid on a peer port. |
| 69 | 45 | The COMMREQ is invalid on a slave port. |
| 70 | 46 | The COMMREQ is valid only on a master port. |
| 71 | 47 | The COMMREQ target ID is invalid. |
| 127 | 7F | Generic miscellaneous error. |

Section 6: CCM COMMREQ Example

This example shows a complete PLC ladder program which sends CCM COMMREQs to the PCM. The program was developed for Series 90-70 PLCs, but it can be used with Series 90-30 PLCs by removing the OK output from the COMMREQ function block in rung 9.

On the first scan, the data for the COMMREQ command block and data block is initialized using the two BLKMOV functions in rung 5. The command block is in %R00050-%R00055. %R00050, the data block length, is 4 since the CCM 6110 command requires 4 words of data. %R00051 contains 0 for **NOWAIT** mode. The CCM status word is returned in %R00107 since %R00052 is 8 for %R memory and %R00053 contains zero-based offset 106. %R00054 and %R00055 are ignored in **NOWAIT** mode and are simply initialized to zero.

The data block begins at %R00056. The first word in the data block is the CCM command 6110, single bit write. This command requires three additional words of data, moved into %R00057-%R00059: target ID, memory type, and target memory address. The target ID is 4, the memory type is 14 for bit set in the output table, and the target memory address is 296. Consequently, the command sets %Q00296 on CCM CPU ID 4.

On the first scan, the program ensures that the PCM has had time to initialize itself before sending the first COMMREQ. This is done with a 5-second TMR block in rung 8. The initial count value for the timer was cleared to 0 by one of the MOVE_UINT function blocks in rung 6. After the timer has expired, %T00001 latches on.

When %T00001 is on, the CCM status word in %R00107 is checked for a non-zero value by the NE_UINT function block. Its value was initialized to 1 on the first scan (by another MOVE_UINT function block in rung 6) to allow the first COMMREQ to be sent. When the status word is not equal to 0, power flow is provided first to a MOVE_UINT function block, which clears the status word, and then to the COMMREQ. Note that the COMMREQ IN parameter specifies %R00050 as the command block location. SYSID is 0002, so the PCM must be located in rack 0, slot 2. The TASK parameter on the COMMREQ indicates that this COMMREQ is for CCM on port 1 of the PCM.

After the COMMREQ completes, a non-zero CCM status word value is moved to %R00107. The NE_UINT block then permits another COMMREQ to be sent.

When the program is run, an error may cause the COMMREQ FT output to become active or cause the PCM to post a fault to the PLC fault table. Either kind of error will prevent the COMMREQ from completing and setting %R00107 to a non-zero value. Additional logic should be provided to detect these faults using PLC fault or system contacts and to recover from them. For more information on Series 90 fault and system contacts, refer to the *Series 90-30/90-20 Programmable Controllers Reference Manual*, GFK-0467, or the *Series 90-70 Programmable Controller Reference Manual*, GFK-0265.

05-04-91 11:05 GE FANUC SERIES 90-70 DOCUMENTATION (v3.01)
 Program to Send CCM COMMREQs As Often As Possible

Page 1

```
[ START OF LD PROGRAM EXAMPLE ]      ( *                               * )
[ VARIABLE DECLARATIONS ]
[ PROGRAM BLOCK DECLARATIONS ]
[ INTERRUPTS ]
[ START OF PROGRAM LOGIC ]

<< RUNG 5 >>

FST_SCN
--] [---|-----|-----|-----|
| BLKMOV |          | BLKMOV |
| INT    |          | INT    |
|-----|          |-----|
CONST - IN1 Q+-%R00057  CONST - IN1 Q+-%R00050
+00004 +00004
CONST - IN2              CONST - IN2
+00014 +00000
CONST - IN3              CONST - IN3
+00296 +00008
CONST - IN4              CONST - IN4
+00000 +00106
CONST - IN5              CONST - IN5
+00000 +00000
CONST - IN6              CONST - IN6
+00000 +00000
CONST - IN7              CONST - IN7
+00000 +06110

<< RUNG 6 >>

FST_SCN
+---] [---|-----|-----|-----|
| MOVE_  |          | MOVE_  |
| UINT   |          | UINT   |
|-----|          |-----|
CONST - IN Q-%R00001  CONST - IN Q-%R00107
+00000 +00001          +00001
| LEN    |          | LEN    |
| 00001  |          | 00001  |
|-----|          |-----|

<< RUNG 7 >>

COMMDLY
(* COMMENT *)
```

Program: EXAMPLE

C:\LM90\EXAMPLE

Block: _MAIN

05-04-91 11:05

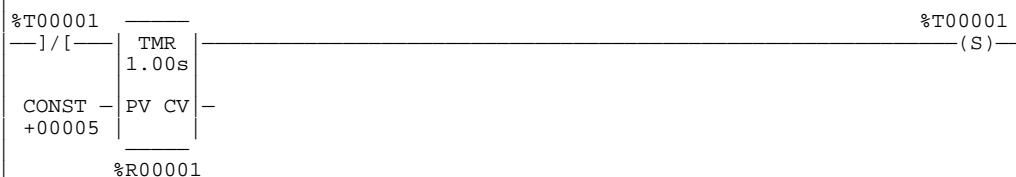
GE FANUC SERIES 90-70 DOCUMENTATION (v3.01)
Program to Send CCM COMMREQs As Often As Possible

Page

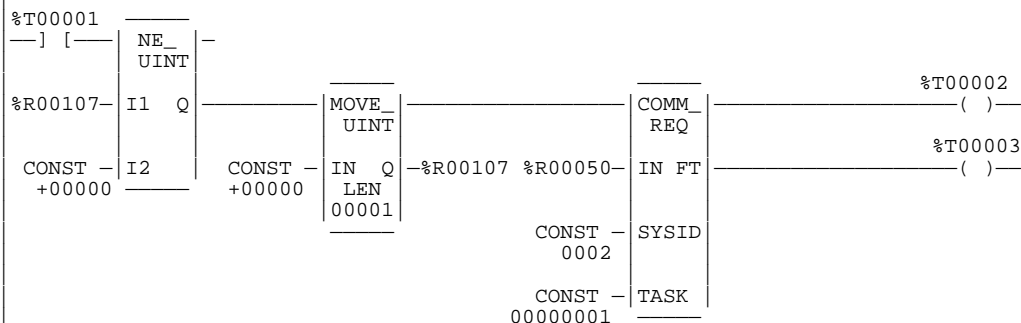
2

```
(*****
(* Delay the first COMMREQ 5 sec.  %T1 is guaranteed to be off when the PLC *)
(* transitions to RUN mode.                                     *)
(*****)
```

<< RUNG 8 >>



<< RUNG 9 >>



<< RUNG 10 >>

COMFLT

(* COMMENT *)

```
(*****
(* Additional rungs should be added to recover when the COMMREQ FT output *)
(* becomes active or when the PCM posts a fault to the PLC fault table. *)
(*****)
```

[END OF PROGRAM LOGIC]

Program: EXAMPLE

C:\LM90\EXAMPLE

Block: _MAIN

Section 7: PLC System Communications Window

Communication between the PLC CPU and the PCM occurs during a portion of the PLC sweep called the **system communications window** or **system window**. In particular, the CPU can receive COMMREQ response messages from the PCM only during the system window.

In Series 90-30 CPUs, the system window time is fixed. The rest of the information in this section applies only to Series 90-70 PLCs.

Series 90-70 System Communications Window

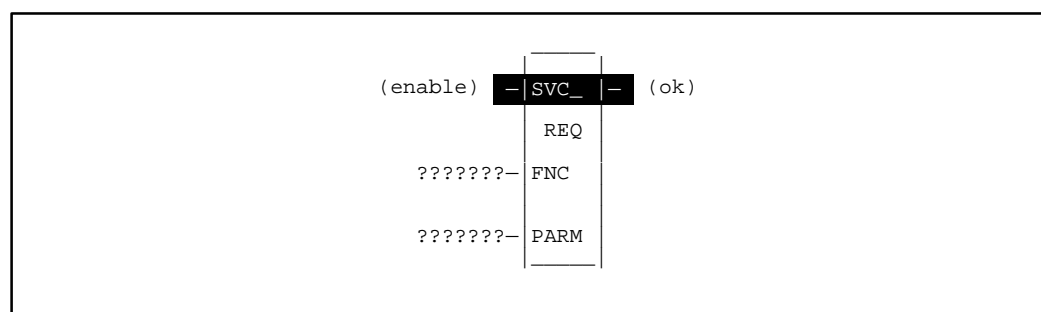
The system window in Series 90-70 PLC CPUs is normally allowed as much time as necessary to process every message to the CPU from every intelligent module in the PLC. This mode of window operation, called **RUN TO COMPLETION** mode, allows the most efficient communication possible. However, some time critical PLC applications require a limit for the system window time.

The system window mode and time in Series 90-70 PLC CPUs can be changed by Logicmaster 90-70 configuration software and by the PLC program. Using Logicmaster 90-70 configuration software is the simpler and faster method. However, if the PLC program must guarantee that the system window is set to the desired configuration, or if the window configuration must change to accommodate different conditions, then the program must configure the system window. The remainder of this section describes how to use the SVCREQ function block to adjust the system window.

PLC Service Request (SVCREQ)

The SVCREQ function block may be used to request a number of services from the PLC CPU. For general information on the SVCREQ function, refer to the *Series 90-70 Programmable Controller Reference Manual*, GFK-0265.

The SVCREQ function block can be used to set the system window as follows:



FNC: Specifies the PLC service to be requested. The value used to set the system window is 4.

PARM: Specifies the memory location of the parameter block. It may be any word-oriented user reference (%R, %AI, %AQ, %P or %L). The parameter block used to set the system window consists of two byte values in a single word:

| High Byte | Low Byte |
|-----------|-------------|
| Mode | Value in ms |

The mode value 0 requests **LIMITED** mode, which permits the window to use as much time as required to process messages, up to the specified limit value. Message processing which is not completed within the time limit is deferred until the next PLC sweep. Time limit values from 0 to 255 milliseconds, in 1 millisecond increments, may be specified. The time limit value 0 prevents the PLC CPU from receiving any messages from PCMs or other intelligent modules. For information on other system window modes, refer to the *Series 90-70 Programmable Controller Reference Manual*, GFK-0265.

The following table shows the minimum system window time values which will not degrade CCM performance, for various CPU models in a PLC which contains one PCM (but no other intelligent modules) configured for CCM on one port:

| Series 90-70 CPU Model | Minimum System Window Time |
|----------------------------------|----------------------------|
| CPU731/732 | 5 ms |
| CPU771/772 | 3 ms |
| CP780 781/782/788/789 | 2 ms |
| CP790 914/924/915/925 | 1 ms |
| CP7X2 782/928/935 | 1 ms |
| CGR772/935 | 1 ms |

The window time limit must be increased when the PCM is configured for CCM on both ports, and when the PLC contains two or more PCMs or other intelligent modules.

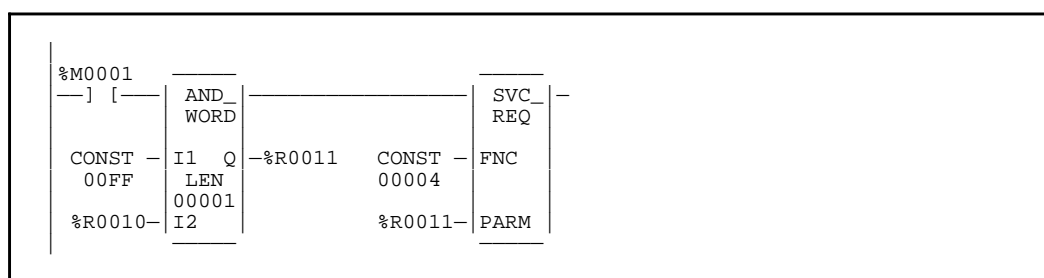
The requirement for fast, predictable PLC sweep times conflicts with the requirement for efficient CCM communication. Different applications will assign different weights to these conflicting requirements. Some applications will emphasize control of sweep time, some will emphasize efficient CCM communication, while others will need a good balance of the two. This complex mix of application requirements, plus the large number of possible combinations of PCMs (and other intelligent modules), makes it virtually impossible to provide a useful rule of thumb for the system window time. Developers of time critical CCM applications will need to tune the system window time for their specific needs.

Series 90-30 System Communications Window

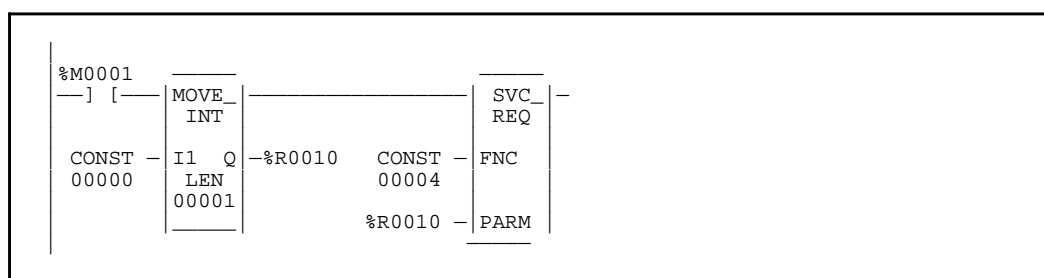
In Series 90-30 CPU versions earlier than 4.4, the system window time is fixed. Starting with version 4.4, however, the default mode of system window operation is **RUN TO COMPLETION** (which is actually limited to 50 milliseconds). PLC programs may use the SVCREQ function block in CPU version 4.4 or later to change the window to **LIMITED** mode. In Series 90-30 CPUs, **LIMITED** mode is fixed at 6 milliseconds.

SVCREQ Examples

The following ladder program rung sets the Series 90-70 system window to **LIMITED** mode and the time limit to the value in the low byte of %R00010 whenever contact %M00001 is active.



This ladder program rung sets the Series 90-30 system window mode to **LIMITED** mode whenever contact %M00001 is active. The window time is fixed at 6 milliseconds.



Chapter 4

MegaBasic

This chapter contains information on developing and running MegaBasic applications in the Series 90 PCM. A PCM with firmware version 2.50 or greater may be configured for BASIC or BAS/CCM operation using Logicmaster 90 configuration software. PCMs with firmware version 2.04 or lower must be configured for BASIC using PCOP, and other PCMs may be configured using PCOP. One of these methods must be used before the PCM can be used for MegaBasic. Refer to chapter 2, section 2, *Configuring the PCM with Logicmaster 90 Software*, for a guide to configuring the PCM.

Chapter 4 contains the following sections:

| Section | Title | Description | Page |
|---------|---|--|------|
| 1 | Programming the PCM in MegaBasic | Section 1 describes the steps required to program a PCM MegaBasic application. | 4-2 |
| 2 | Interfacing to the PCM Hardware and Series 90 CPU | Section 2 covers PCM serial port and PLC data access from MegaBasic. | 4-13 |
| 3 | MegaBasic Programming Examples | Section 3 contains example MegaBasic programs. | 4-26 |

Section 1: Programming the PCM in MegaBasic

MegaBasic is a powerful BASIC language interpreter which is built into the PCM. MegaBasic programs can be created using:

- Any VT100-compatible terminal.
- TERMF, the PCM support software, running in a Workmaster industrial computer or IBM PC-XT, PC-AT or PS/2 personal computer. Many IBM-compatible personal computers can also run TERMF.
- PCOP, the PCM development software, running in one of the personal computers described above.

MegaBasic programs can be saved in PCM user RAM; they can also be loaded and saved to a Workmaster computer or compatible PC by using TERMF or PCOP.

MegaBasic on the PCM provides backplane access to Series 90 PLC CPUs and other features to support process control and real-time programming. These MegaBasic **extensions** developed by GE Fanuc, are also built into the PCM and are automatically accessed by all MegaBasic programs.

PCM MegaBasic programs can be developed off-line, using any editor that creates ASCII text files. However, programs created off-line can be executed only when installed in the PCM.

A separate version of the MegaBasic interpreter allows you to create and execute programs in a DOS-based personal computer. The MS-DOS version of MegaBasic, however, does not provide the PCM extensions. It is available from GE Fanuc as catalog number IC690SHP403 (for 3.5-inch diskettes) or IC690SHP404 (for 5.25-inch diskettes).

The remainder of this chapter describes how to use MegaBasic in the PCM. For information on MegaBasic commands and the MegaBasic language itself, refer to the *MegaBasic Programming Language Reference Manual*, GFK-0256.

Getting Started with the MegaBasic Interpreter

Before the PCM can be used for MegaBasic applications, it must be configured. If your PCM has firmware version 2.50 or greater, you can use Logicmaster 90 configuration software to configure the PCM for MegaBasic only (**BASIC** mode) or for simultaneous MegaBasic and CCM operation (**BAS/CCM** mode). Refer to chapter 2, section 2, *Configuring the PCM with Logicmaster 90 Software*, for information on **BASIC** mode configuration.

The PCM can also be configured and programmed for MegaBasic using the PCM development software (PCOP). PCMs with firmware version 2.04 or lower must use PCOP. Refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487, for information on configuration and programming the PCM using PCOP.

When the PCM is configured by Logicmaster 90 for **BASIC** or **BAS/CCM** mode, a hard reset (pressing the Restart/Reset pushbutton for 10 seconds) places the PCM programming port at the MegaBasic command level. If your PCM is connected to an ASCII terminal or to a personal computer running TERMF, you can enter MegaBasic commands and program lines. If you are using TERMF, you can load an existing program from a file on the PC.

A soft reset (pressing the Restart/Reset pushbutton for less than 5 seconds) will cause MegaBasic to load and run a user program with the reserved name **BASIC.PGM**, if one is present in PCM RAM. If there is no user program named **BASIC.PGM**, MegaBasic behaves as if it had received a hard reset.

To type new MegaBasic program lines, enter a line number followed by one or more program statements, separated by semicolons. Use a carriage return to terminate the program line. Lines may be up to 254 characters long. New lines may be entered in any order, regardless of the line number. The line number simply tells MegaBasic where to insert the new line in the current program.

Loading and Saving MegaBasic Programs

The default device for PCM file locations is RAM:, the PCM RAM Disk. File access, using the MegaBasic **LOAD** and **SAVE** commands or the **ACCESS** statement, assumes the RAM Disk unless another device is explicitly specified. On any file access, the device name may be added in front of the file name. For example, **RAM:BASIC.PGM** fully specifies a file called **BASIC.PGM** on the PCM RAM Disk. **PC:BASIC.PGM** is a file located in the current directory of the current PC disk drive.

To load an existing MegaBasic program file to the PCM MegaBasic workspace from the default device, type **LOAD <filename>**, where **<filename>** is the name of the program file.

Add a device name to explicitly specify the PCM RAM: device or the current PC disk drive. For PC files, you can also specify the MS-DOS file directory path:

LOAD <device>:<path> <filename>, where **<device>** is the storage device, **<path>** is directory path to the program, when applicable, and **<filename>** is the program name. You can also specify a PC disk drive by using two device names. For example:

| Path | Description |
|--|--|
| LOAD MYPROG.PGM | Current directory of default device. |
| LOAD RAM:MYPROG.PGM | PCM RAM Disk. |
| LOAD PC:MYPROG.PGM | Current directory of current PC disk drive. |
| LOAD PC:\MYPROG.PGM | Root directory of current PC disk drive. |
| LOAD PC:C:MYPROG.PGM | Current directory of PC drive C. |
| LOAD PC:\MB\MYPROJ\MYPROG.PGM | Directory \MB\MYPROJ of current PC disk drive. |
| LOAD PC:C:\MB\MYPROJ\MYPROG.PGM | Directory \MB\MYPROJ of PC drive C. |

Note that there are no paths on the PCM RAM: device.

To save a MegaBasic program to default device and directory, type **SAVE <filename>**. You can also save programs explicitly to the PCM RAM Disk or a specified PC disk drive and/or file path:

| Path | Description |
|--|--|
| SAVE MYPROG.PGM | Current directory of default device. |
| SAVE RAM:MYPROG.PGM | PCMRAMDisk. |
| SAVE PC:MYPROG.PGM | Current directory of current PC disk drive. |
| SAVE PC:\MYPROG.PGM | Root directory of current PC disk drive. |
| SAVE PC:C:MYPROG.PGM | Current directory of PC drive C. |
| SAVE PC:\MB\MYPROJ\MYPROG.PGM | Directory \MB\MYPROJ of current PC disk drive. |
| SAVE PC:C:\MB\MYPROJ\MYPROG.PGM | Directory \MB\MYPROJ of PC drive C. |

If you want a program to run automatically after power is applied to the PLC or a PCM soft reset occurs, save the program to RAM: and use the file name **BASIC.PGM**.

When you load a program into a new workspace, MegaBasic remembers where it was loaded from and uses the same device and file name as defaults when the program is saved. For example, if you type **LOAD PC:<filename>**, make some changes, and then **SAVE** the program, it is saved to **PC:<filename>**. You can specify a new file name on the default device. For example, typing **SAVE <newfilename>**, would save the program to **PC:<newfilename>**. You can also specify a new device for the default file name or a new device with a new file name. Typing **SAVE RAM:** or **SAVE RAM:<newfilename>** saves the program to the RAM Disk.

For most file access commands, MegaBasic prompts you for confirmation. The confirmation prompt includes the destination device before the file name, providing a check to verify that the device you intend will actually be used. To ensure that the correct destination for a file access command will be used, you can always fully qualify file names with the device name.

Backing Up Your Program

If you perform a power cycle or reset the PCM while adding lines to a MegaBasic program, the program statements entered in the current session are lost. Therefore, when you develop a program on-line, you should back it up to a PC: file often.

Although use of the RAM Disk for program storage is much faster than using the PC, programs stored on the RAM Disk can be corrupted under certain conditions. In particular, if a MegaBasic program uses assembly language routines or the **FILL** statement, the program and data files should always be backed up to PC: before running. It is good programming practice to back up every change to programs and data files to the PC hard disk.

Exiting the MegaBasic Interpreter

Most MegaBasic programs developed for the PCM are programmed to run in a continuous loop and are restarted only when the PCM is reset. During debug, however, it may be necessary to stop a program. There are three ways a MegaBasic program can be stopped. If a syntax error occurs, CTRL-C is pressed, or a **STOP** statement is encountered, MegaBasic stops and returns to **PROGRAM DEVELOPMENT** mode. Pressing the Enter key displays the last few lines that were executed.

If a MegaBasic program terminates instead of looping continuously, or if an **END** statement is encountered, there are two possible results. If the program was started from MegaBasic in **PROGRAM DEVELOPMENT** mode, MegaBasic returns to **PROGRAM DEVELOPMENT** mode just as if it were stopped. If the program was run automatically (by configuring the PCM with Logicmaster 90 software, saving the program to **RAM: BASIC.PGM** and resetting the PCM), however, MegaBasic itself will exit. This also happens if you type **BYE** from the MegaBasic command line.

If your PCM was configured using Logicmaster 90 software, and you type **BYE** or press the Enter key after the PCM has completed running a MegaBasic program, the ">" character is displayed. At this point, you are no longer communicating with MegaBasic. A hard reset (pressing the Restart/Reset pushbutton for 10 seconds) returns the PCM to MegaBasic **PROGRAM DEVELOPMENT** mode.

If MegaBasic was started with PCOP, exiting MegaBasic will return you to a PCOP menu. Refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487, for details.

Saving Data through a Power Cycle or Reset

When the PCM is reset, the MegaBasic program is restarted and all of its variables are initialized to zero. If data must be saved through a power failure, it can be written to a RAM Disk file on the PCM. RAM Disk files are always saved through power-up. When the program starts up, it should open the RAM file, check to see if it contains data from the previous run, and take the appropriate action. For details on creating and using MegaBasic files, refer to the *MegaBasic Programming Language Reference Manual*, GFK-0256.

Compatibility with MS-DOS MegaBasic

MegaBasic is also available in an MS-DOS version, which may be ordered from GE Fanuc as catalog number IC690SHP403 (for 3.5-inch diskettes) or IC690SHP404 (for 5.25-inch diskettes). This version consists of the *MegaBasic Programming Language Reference Manual*, GFK-0256, and diskettes containing the software. The PCM extensions to MegaBasic are not included in the MS-DOS version. You can develop MegaBasic programs which run in either the PCM or an MS-DOS computer if you avoid the PCM extensions. A good technique is to isolate all the PCM extensions within a single MegaBasic package. A different version of the package, using the same package name but none of the PCM extensions, can be substituted when the program is run in a MS-DOS computer.

Programs and data files for the PCM version of MegaBasic are compatible with the MS-DOS version. The one difference is the method used to specify files that are not on the current PC disk drive. In the MS-DOS version, you can specify a disk drive for files in the device portion of the file path. For example:

| Path | Description |
|------------------------------|---|
| MYDATA.DAT | Current directory of the current drive. |
| \MB\MYPROJ\MYDATA.DAT | Directory \MB\MYPROJ of the current drive. |
| A:MYDATA.DAT | Current directory of drive A. |
| C:\MB\MYDATA.DAT | Directory \MB of drive C. |

In the PCM version, however, the usual device part of the file path is used to distinguish between the PCM RAM Disk and the current disk drive in the PC. Consequently, a second device must be specified in the path of PC files which are not in the current drive:

| Path | Description |
|---------------------------------|--|
| RAM:MYDATA.DAT | PCMRAMDisk. |
| PC:MYDATA.DAT | Current directory of the current PC drive. |
| PC:\MB\MYPROJ\MYDATA.DAT | Directory \MB\MYPROJ of the current PC drive. |
| PC:A:MYDATA.DAT | Current directory of PC drive A. |
| PC:C:\MB\MYDATA.DAT | Directory \MB of PC drive C. |

You can also hide this difference between the PCM and MS-DOS versions of MegaBasic in the single package which you change when switching between them. Define string variables in this package for all file path and name specifications, and assign different string values to them in the PCM and MS-DOS versions of the package. Then, use the string variables as arguments in all MegaBasic statements and functions which specify the files by name.

Unlike PC disk drives in DOS version 2.0 or later, the PCM RAM Disk does not support multiple directories.

MegaBasic Features Not Supported by the PCM

Background processing and network support are not provided by the PCM version of MegaBasic, nor is debug mode screen switching. The time and date may be read but not changed, since clock synchronization is maintained with the PLC CPU. Also, there are no environment string processing routines for the PCM version of MegaBasic.

In addition, the following statements and functions are not supported in the PCM:

| Command | Description |
|-----------------------|---|
| Rename | Change the name of a file. |
| DOS (with parameters) | Execute OS shell-level command from the user program. |
| Param(2) | Param function, which resets the disk drive. |
| Filedate\$ | Return date file last modified. |
| Filetime\$ | Return time file last modified. |
| Dir\$ | Display or change current directory. |
| Subdir\$ | Returnsubdirectory names in a directory. |
| Envir\$ | Access DOS environment strings. |
| Ioctl | Test if device supports IOCTL strings. |
| Space | Return disk space. |
| Free | Return a portion of memory to the operating system. |

The **IOCTL\$()** function can be used to send a configuration string to the PCM serial ports COM1: and COM2:, as described in chapter 5, *Advanced MegaBasic Programming*, but it cannot be used for any other device. Furthermore, **IOCTL\$** cannot be used to return input strings from any device.

The **DIR** command can be used to display the directory contents of various PCM devices, but it cannot be used to change the default directory. Type **DIR** to display the PCM's RAM directory. To display the current directory for an attached PC, type **DIR "PC:"**.

Finally, when using the PCM version of MegaBasic, device names must include a colon. For example, to open the second serial port, type **OPEN #5, "COM2:"**. The colon is optional with the MS-DOS version.

Modifying Existing BASIC Programs for MegaBasic

It is possible to load many programs developed for other BASICs directly into MegaBasic. Using either the **CHECK** or **RUN** command will cause the MegaBasic interpreter to indicate any instructions that are not supported. Lines with unsupported statements or functions can be replaced with equivalent MegaBasic statements or functions. While this is a quick way to convert a program to MegaBasic or estimate the size of an application, it is not recommended for the final application because it does not take advantage of the strength and efficiency of programs written specifically for MegaBasic.

Printing a MegaBasic Text File

When a PCM MegaBasic program is saved, it is stored in a special format. It can be printed when in this format only by using the MegaBasic **LIST** command, which by default prints it to the screen.

To print the program or obtain the text version for editing off-line, you must **LIST** it to a PC file. First, invoke MegaBasic and load the program. Then, type **OPEN #5, "PC:filename.lst"** to create an output file. Any unused channel number can be used instead of 5, and you can, of course, use any file name instead of **filename.lst**. When MegaBasic responds with the "Ready" prompt, type **LIST #5** to list the program to the PC file specified by the file name. Once this is done, you can exit MegaBasic and use the MS-DOS **PRINT** command to print the file or a text editor program to edit it. If you do not exit MegaBasic right away, type **CLOSE #5** before continuing.

Using a Text Editor to Create MegaBasic Programs

Any text editor that creates or uses ASCII text files can be used to create or modify MegaBasic programs. To create a MegaBasic program, simply type MegaBasic statements. Line numbers are not required and are added during the automatic text file conversion process in MegaBasic. Save the text file to disk, and use it as any other MegaBasic program.

To edit an existing MegaBasic program on your PC, list the program to a file, following the instructions above. The resulting file may be edited.

When loading the text file into MegaBasic, the message "Text file conversion" is displayed. You can check the syntax of your program before running it by using the **CHECK** command.

MegaBasic Program and Data Size

MegaBasic is more efficient than many other BASIC interpreters because it uses structured programming concepts to minimize program size. Conversion of a program originally written for a different BASIC "dialect" to MegaBasic often reduces its size significantly, especially if the powerful MegaBasic features are used. A good rule of thumb for estimating the size of a typical MegaBasic application intended for an operator interface terminal is 50K bytes plus 4K bytes per screen.

When MegaBasic executes a user program, it must first load the program into an internal workspace. During execution, the program is continually optimized to provide the best possible performance. In order to restart the application following a reset, the original copy of the program must be stored on the PCM RAM Disk. This reduces the effective amount of memory available for storing programs on the PCM, since the PCM must hold both a working copy and an original copy of the program.

When MegaBasic starts up, it takes part of the PCM memory for program workspaces and leaves the rest of memory for the PCM RAM Disk. When you **LOAD** a program into MegaBasic, it is copied from the RAM Disk (or from your PC) into a program workspace. When you **SAVE** a program, it is copied from program workspace to the RAM Disk (or your PC).

The division of PCM memory between RAM Disk and MegaBasic workspace is performed automatically when the PCM is configured by Logicmaster 90 software for **BASIC** or **BAS/CCM** mode. PCOP configuration also allocates PCM memory between RAM Disk and MegaBasic workspace. In addition, PCOP allows you to modify the workspace allocation. Refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487, for information on using PCOP to configure the amount of PCM memory allocated to MegaBasic.

The following table lists the maximum sizes for RAM Disk and program workspace, corresponding to the various PCM memory configurations. Note that the program workspace is larger than the RAM Disk space because the program workspace must contain executing programs, data (variables used by MegaBasic), and the PCM extensions, while the RAM Disk needs to hold only the MegaBasic program files.

Table 4-1. Default Program Workspace and RAM Disk Sizes

| <i>Series 90-70 PCM</i> | | | |
|-------------------------------|------------------|--|----------------------------------|
| RAM Sizes | | Default Mega Basic Workspace Size | Default RAM Disk Size |
| Option RAM | Total RAM | | |
| None | 128K | 48K | 20K |
| 64K | 192K | 86K | 45K |
| 128K | 256K | 133K | 60K |
| 256K | 384K | 213K | 105K |
| 512K | 640K | 374K | 199K |
| <i>Series 90-30 PCM</i> | | | |
| RAM Size Total RAM | | Default MegaBasic Workspace Size | Default Disk Size |
| 160K | | 66K | 30K |
| 192K | | 86K | 45K |
| 640K | | 374K | 199K |

Determining the Size of a MegaBasic Program

The MegaBasic **STAT** command can be used to determine the size of a program's code and data, as well as the remaining workspace size. The value for bytes remaining is the unused workspace available for program and data at that time. The **SHOW** command prints the program and data sizes of all programs loaded into MegaBasic workspace.

MegaBasic Program Packages

MegaBasic programs may be partitioned into packages. The main program can execute one or more **ACCESS** statements to load additional packages into the MegaBasic workspace. This feature permits a logically related collection of variables, procedures, and functions to be grouped together in a package. The main program can then have a more compact, easily understood structure.

The cost of storing multiple packages in the PCM is at most a few bytes per package. The RAM Disk space needed to store several smaller packages is about the same as the space for the equivalent larger one.

The length of MegaBasic packages, including the main program, is limited to 64K bytes. MegaBasic complains if you attempt to load a longer program in either ASCII text or processed format. Consequently, programs larger than 64K bytes must be divided into two or more packages.

Only the main program and the package which is currently executing need to be in MegaBasic workspace. The **DISMISS** statement can remove a package from the workspace when its work is done, making room for another package. This technique permits significantly larger PCM applications because all the code does not have to be in MegaBasic workspace at once.

For more information on packages, see the *MegaBasic Programming Language Reference Manual*, GFK-0256.

Changing the MegaBasic Workspace Size

Under certain conditions, it may be desirable to change the division of the PCM memory between MegaBasic workspace and RAM Disk. For example, if the MegaBasic program is fairly small but uses a lot of data, you may want to use less memory for the RAM Disk, which only needs to hold the program, and use the remainder of PCM memory for MegaBasic workspace.

If your PCM has firmware version 2.50 or greater, you may change the workspace size for MegaBasic in a **PCMEXEC.BAT** file. See appendix D, *PCM Batch Files*, for information on PCM batch files. See appendix C, *PCM Commands*, for information on using the /D option of the **R (Run)** command to assign a workspace size.

The MegaBasic workspace size can also be changed using the PCM development software, PCOP. Refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487, for information on using PCOP to allocate PCM memory.

Compacting and Encrypting Programs

CRUNCH.EXE is a program compaction utility. It is supplied with both TERMF and PCOP software. It is installed in the `\PCOP\UTILS` directory on the PC hard disk for TERMF and in the `\PCOP` directory for PCOP. The program is run from the MS-DOS prompt to process programs in PC files. It removes comments and extra spaces, including indentation, from programs. CRUNCHED programs can be LISTed and modified, but they are far less readable than unCRUNCHED programs.

CRUNCH is useful for squeezing large applications into PCM memory. Fully commented programs with indentation are often more than twice as large as their CRUNCHED versions. During development, you can save the unCRUNCHED version of your application to your PC and keep only the MegaBasic workspace copy on the PCM. Then, when development is complete, CRUNCH the application and save it to the PCM RAM Disk. The total PCM memory used by both the RAM Disk and MegaBasic workspace copies of the CRUNCHED version will be about equal to the workspace used by the unCRUNCHED version.

CRUNCH does not work with MegaBasic programs in ASCII text file format. Before you can CRUNCH an ASCII text program, you must load it to MegaBasic and then save it to the PC.

The syntax of the command is **CRUNCH** `<filename>` `<newfilename>`, where `<filename>` is the unCRUNCHED program and `<newfilename>` is the CRUNCHED version (for example, **CRUNCH** old.pgm new.crn). **CRUNCH** prompts you for various option choices.

CRUNCH.EXE has an encryption option, which uses a cipher or scrambling technique to prevent unauthorized users from reading or modifying programs. This feature can be used with PCM MegaBasic programs; however, the main program may not be encrypted. A CRUNCHED or unCRUNCHED main program can access subprograms or packages that are CRUNCHED or CRUNCHED and encrypted.

Caution

Make sure that you save your source program in a safe place before CRUNCHing or encryption. There are no unCRUNCH or decryption utilities.

CRUNCHED and encrypted MegaBasic packages cannot be loaded to MegaBasic workspace using the MegaBasic **LOAD** command. They must be loaded to the PCM RAM Disk using TERMF or PCOP. See chapter 5, section 6, *Loading and Storing PCM Data Files Using TERMF*, for more information on using TERMF to load files to the PCM.

Section 2: Interfacing to the PCM Hardware and Series 90 CPU

MegaBasic has access to all the PCM input/output devices. Through Release 3, these include the RAM Disk (RAM:), two PCM serial ports (COM1: and COM2:) and a file server (PC:), which can be attached to either serial port. Finally, there is a NULL device (NULL:), which can be used to discard output.

Note

When using the PCM version of MegaBasic, device names must include a colon. The colon is optional with the MS-DOS version.

MegaBasic programs can perform input and output (I/O) operations using either **standard** or **specified** devices. MegaBasic is normally configured to use COM1: as its input and output device. For example, executing the line:

```
10 Print "Hello World"
```

causes the string "Hello World" to be printed out port 1 of the PCM.

To read or write one of the other devices, the program must first open the device and assign it a specified channel number. This channel number is then used in subsequent I/O statements. For example, the following two lines:

```
10 Open #5, "COM2:"  
20 Print #5 "Hello World"
```

print the string "Hello World" to serial port 2.

In the MS-DOS version of MegaBasic, channel numbers 0, 1, and 2 are reserved for the console, printer, and auxiliary device, respectively. In the PCM version of MegaBasic, the channel numbers 0, 1, and 2 are assigned to the logical devices standard input, standard output, and standard error, respectively. They are set to COM1 by default, although they can be changed, if necessary, with a PCM batch file or PCOP. Changing them is required, for example, to prevent the MegaBasic startup banner from appearing on a display terminal attached to the PCM. For information on changing the port assigned to standard channels, refer to the **R (Run)** command in appendix C, *PCM Commands*, in this manual or to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487.

Channel numbers 3 and 4 are reserved for future use by GE Fanuc. Your programs should assign channel numbers from 5 through 31 to avoid possible conflicts with future PCM releases.

Input and Output to the PCM Serial Ports

The PCM has two high-performance serial ports that can be used to connect the PCM to any device that uses the RS-232, RS-422, or RS-485 physical connection protocol. The COM1: and COM2: device drivers, which come with the PCM, support asynchronous communication at data rates up to 38.4K bits per second.

There are several ways to input characters from a serial port to a MegaBasic program. The simplest (although not the recommended) method is the MegaBasic **INPUT** statement. When an **INPUT** statement is executed, the MegaBasic program waits until the Enter key is pressed. All the keys pressed, including the Enter key, are collected and echoed to the screen as they are pressed. You can delete and retype characters. The **INPUT1** and **INPUT2** statements are similar to **INPUT**, except that **INPUT1** suppresses the Enter key echo and **INPUT2** suppresses all character echoing and editing.

Because the **INPUT** statements wait forever if the Enter key is not pressed, they should never be used in applications which need to perform other functions on a regular time schedule. The MegaBasic **INCHR\$** function is recommended for these applications; it can be programmed to time out when no input is typed. It can also be programmed to return its input after a specified number of characters has been typed or when any one of a specified set of terminating characters is typed.

The **NOWAIT_IO** functions, developed for MegaBasic by GE Fanuc, are also recommended. These functions provide the capability for interrupt-driven I/O from MegaBasic. For information on these functions, refer to chapter 5, section 9, *COMMREQs and Other Backplane Messages*.

Other MegaBasic I/O statements and functions which are not recommended are **INP**, **INP\$**, and **OUT**. These require direct access to the PCM serial hardware. Using them can seriously interfere with the operation of higher level MegaBasic I/O statements and functions as well as CCM, which can operate simultaneously with MegaBasic.

Most program output is done with the MegaBasic **PRINT** statement. It provides a large number of formatting capabilities. **TERMF** and **PCOP** provide several utilities that are useful for writing to terminals, such as routines to clear the screen and position the cursor. Refer to chapter 5, *Advanced MegaBasic Programming*, for information on these utility programs.

The **PRINT** statement always waits until output is complete before continuing. The **NOWAIT_IO** functions, described in chapter 5, provide buffered output, which allows the program to continue while output occurs.

Serial Port Control and Status

MegaBasic has several built-in statements and functions for controlling devices and determining their status. The two status functions are **INPUT()** and **OUTPUT()**. These are used to determine the input status and output status, respectively, of any device. The **INPUT()** function is particularly useful for determining when input data is available from an operator or external device.

Accessing PLC Data

Because the PCM and Series 90 PLC CPU are connected by the PLC backplane, PCM MegaBasic programs can transfer data between the PCM and the PLC CPU very efficiently. In order to transfer data, a MegaBasic program must first create an association between a MegaBasic variable and an area in the PLC memory. This is done with the **SYSLINK** statement. Once a MegaBasic variable has been associated with PLC data, the PLC data can be copied to or from the MegaBasic variable using the **SYSREAD** and **SYSWRITE** statements. The **SYSTATUS\$** function is used to monitor data transfers between the PCM and PLC. Each of these is described in detail on the following pages.

The maximum number of PLC data areas that can be associated with MegaBasic variables at one time is 32. PLC data areas can be as small as a single point. In Series 90-70 PCMs, the size of each PLC data area is limited to 2048 bytes. In principle, Series 90-30 PLC data areas can be as large as 65,535 bytes, although there were no data areas that large when this manual went to press. MegaBasic arrays or structures can be linked to move multiple data references from or to the PLC CPU in one **SYSREAD**/**SYSWRITE** operation, as shown in the following example.

This example assumes that PLC registers 250 through 329 contain 16 character ASCII formatted real numbers. The numbers are read from the PLC CPU, converted from ASCII to binary format, and written back to the PLC, starting at register 1000.

```

10 Rem This program reads a large string of ascii numbers and
20 Rem converts them to individual real elements of an array.
30 Dim STR1$(160)
40 Dim real REAL_ARR(9)
50 SYSLINK STR1$, "%R250"
60 SYSLINK REAL_ARR, "%R1000"
70 SYSREAD STR1$
80 I% = 0
90 While I% < 10
100 REAL_ARR(I%) = val(STR1$(I%*16+1:16))
110 Print REAL_ARR(I%)
120 I% = I%+1
130 Next
140 SYSWRITE REAL_ARR
150 Print STR1$

```

More than 32 individual PLC data areas can be associated with MegaBasic variables, if necessary, by using the **UNLINK** statement to disassociate PLC data areas not needed immediately.

A MegaBasic program can read and write any of the PLC data areas shown below using the **SYSLINK**, **SYSREAD** and **SYSWRITE** statements. The names for the areas are shown as they appear in the **SYSLINK** statement:

```

%IXXXXX - Input contacts
%QXXXXX - Output coils
%MXXXXX - Internal coils
%TXXXXX - Temporary coils
%SXXXXX - System status
%RXXXXX - CPU registers
%GXXXXX - Genius global data
%AIXXXXX - Analog inputs
%AQXXXXX - Analog outputs

```

Note

For information on how to access Series 90-70 %P and %L references from PCM MegaBasic programs, refer to chapter 5, *Advanced MegaBasic Programming*.

In addition, the PCM can read fault information for %I, %Q, %AI and %AQ references; override information for %I, %Q and %M references; and, in Series 90-70 PLCs, transition information for %I, %Q, %M, and %T references. To access this information, a comma followed by an F, O, or T (for fault, override, or transition, respectively) is added to the end of the reference name.

For example, the fault bit for input 39 is accessed as “%I39,F”. The transition bit for temporary coil 15 is accessed as “%T15,T”. Fault, override, and transition bits can be accessed as single bits or groups of bits. They can only be read by the PCM. Any attempt to write to these areas results in an error.

PLC data references may be accessed as single items or arrays. Arrays containing up to 64K bytes of contiguous PLC data may be transferred using a single MegaBasic statement. At least 256 bytes of large arrays are transferred during each PLC execution sweep.

Data transfers between the PLC CPU and a PCM are most efficient when they contain 256 bytes or less. PLC data which is accessed by PCM programs should be collected into contiguous groups when the PLC program is developed.

There is a PLC status variable, called **#SSTAT**, which may be read by the PCM. Through this status variable, the PCM can find out information about the PLC state, sweep time, and various other status information. The **#SSTAT** variable is described in chapter 5, *Advanced MegaBasic Programming*.

PLC nicknames are not directly accessible from the PCM.

SYSLINK

The **SYSLINK** statement is used to identify variables located in the PLC CPU. **SYSLINK** tells the PCM to establish a link with the CPU, allowing the PCM to access user reference data in the CPU. Before any PLC data can be read or written by the PCM, it must be SYSLINKed.

The **SYSLINK** statement has the form:

```
SYSLINK <local_name>, <cpu_symbol>, [type], [handle]
```

| Argument | Description |
|-----------|--|
| LocalName | This argument contains the name of the MegaBasic variable to be associated with the PLC data. The MegaBasic variable must be defined in the MegaBasic program before it can be SYSLINKed. It can be a string of any length, an integer, a real, or a numeric array of up to 3 dimensions. String arrays, sub-strings, and individual elements of arrays cannot be SYSLINKed. |
| CPUSymbol | This argument contains the PLC reference with optional suffix for override, transition, or fault data, as described above. It can be either a quoted string (e.g., "%R500") or a string variable that contains a properly formatted CPU reference. |
| Type | <p>This argument contains the MegaBasic type of the CPU reference, as it exists in the PLC. This type may differ from the type of the local variable defined in the MegaBasic program. For example, a MegaBasic local variable may be defined as an integer, which in MegaBasic is a 32-bit number. If the corresponding CPU variable is a register, which is a 16-bit number, the type argument can be used to convert it to 32-bits when it is read by the MegaBasic program.</p> <p>If a type is specified, the PCM automatically performs a type conversion between the CPU type and the local type whenever the variable is read or written. MegaBasic has three "native" data types, 32-bit integers, 64-bit real numbers, and strings. The following PLC CPU data types may be specified:</p> <ul style="list-style-type: none"> • BOOL = single bit. • BYTE = 8 bits. • INT16 = 16-bit signed integer. • UINT = 16-bit unsigned integer. • DINT = 32-bit signed integer. • REAL32 = 32-bit IEEE real number. • REAL64 = 64-bit IEEE real number. <p>If the type argument is omitted, the PCM assumes that the PLC reference is in the same format as the local MegaBasic variable (32-bit integer, 64-bit real number, or string). For numeric data, this is usually not true since the CPU and MegaBasic have different numeric data types. However, there may be some circumstances (e.g., one or more PCMs moving data through the CPU) where numeric data does not have to be converted.</p> |
| Handle | This argument contains an arbitrary, user-specified integer used when backplane interrupts are enabled. (For more information on backplane interrupts, refer to chapter 5, <i>Advanced MegaBasic Programming</i> .) A MegaBasic program may be interrupted when backplane transfers occur, rather than waiting for each transfer. The handle is used to identify the variable that was transferred. If the handle argument is used, the type argument must also be present, because arguments can be omitted only from right to left. |

Example:

In the following example, **SYSLINK** is used to associate a MegaBasic variable called **PUSHBUTTON** with input %I100. The variable **TEMPERATURE** is associated with register %R25:

```
110 Def integer PUSHBUTTON
120 Def integer TEMPERATURE
130 SYSLINK PUSHBUTTON, "%I100", BOOL
140 SYSLINK TEMPERATURE, "%R25", UINT
```

When the MegaBasic program does a **SYSREAD** from the **PUSHBUTTON** variable, the PCM reads %I100, converts it to an integer, which is either 0 or 1, and copies it to the **PUSHBUTTON** variable.

When the MegaBasic program does a **SYSREAD** from **TEMPERATURE**, the PCM reads %R25 and converts it to an integer between 0 and 65,535. If %R25 is specified as **INT16** rather than **UINT**, the PCM converts it to a number between -32,768 and 32,767.

When the MegaBasic program does a **SYSWRITE** to **PUSHBUTTON**, it sets %I100 if **PUSHBUTTON** is non-zero. If **PUSHBUTTON** is equal to zero, %I100 is cleared. A **SYSWRITE** to **TEMPERATURE** copies the least significant 16 bits of the **TEMPERATURE** variable to %R25.

Example:

Another useful application of MegaBasic's automatic type conversion is to convert PLC integers to MegaBasic real numbers. The MegaBasic real numbers can then be operated on with real number arithmetic and copied back to the PLC as integers. For example:

```
110 Def real CURRENT
120 SYSLINK CURRENT, "%R45", UINT
```

A **SYSREAD** from **CURRENT** converts %R45 to a real and copies it to the **CURRENT** variable. It can now be used in real number expressions with no loss of accuracy. A **SYSWRITE** to **CURRENT** copies the integer part back to %R45.

Using the SYSLINK Statement

It is best to put all the **SYSLINK** statements at the beginning of a MegaBasic program, immediately after the variable declarations, or in the prologue section of a package. This makes the program easier to read and provides a way to document the different data transfers done with the PLC. The only exception to this rule occurs when a single MegaBasic variable is alternately **SYSLINKed** and **UNLINKed** to different PLC data areas during program execution. This should be avoided, if possible.

When a CPU location is referred to, the size of the CPU data object is determined automatically from the size of the MegaBasic local variable and the type argument of the **SYSLINK** statement. This is done automatically by the PCM.

For numeric data, the size of the CPU data object is equal to the number of elements of the MegaBasic local variable (if it is an array variable), multiplied by the size specified by the type argument of the **SYSLINK** statement. A single MegaBasic integer or real variable SYSLINKed with a type argument of Boolean would refer to a single bit in the PLC CPU. An array of 5 integers or reals SYSLINKed with type Boolean would refer to 5 bits. An array of 16 MegaBasic integers or reals SYSLINKed with type UINT would refer to 16 contiguous 16-bit locations or 256 bits.

For string data, the size of the CPU data object is equal to the maximum size of the string in bytes. MegaBasic keeps two types of size information for each string: current size and maximum size. When strings are first created, their current size and their maximum size are equal to their dimensioned size. When an assignment is made to the string, its current size changes according to the assignment that is made. However, when the string is transferred using **SYSREAD** or **SYSWRITE**, the number of bytes in its maximum size is transferred. Furthermore, whenever a string is used as a **SYSREAD** variable argument, its current size is set equal to its maximum size.

Caution

Once a MegaBasic array or string variable has been SYSLINKed, it must not be redimensioned until it is UNLINKed, or erratic behavior results. Although MegaBasic allows redimensioning, it is not a good programming practice and should be avoided.

If the variable to be passed between the PLC and the PCM is a MegaBasic array variable, the type argument can be used to convert all elements of the array from one representation to another. For example, a MegaBasic program could use the type argument to convert 10 consecutive input points at %I300 to an array of ten MegaBasic integers as follows:

```
1000 Dim integer CPU_INPUTS ( 9 )
1010 SYSLINK CPU_INPUTS, "%I300", BOOL
```

The program could now transfer the variables using the statement:

```
1030 SYSREAD CPU_INPUTS
```

This sets each of the ten integers in CPU_INPUTS to 1 or 0, corresponding to the bit values of the ten CPU inputs in %I300 through %I309.

SYSREAD, SYSWRITE, and SYSTATUS\$

To reference a Series 90 PLC CPU variable, the PCM MegaBasic program must define the variable locally and SYSLINK it to a CPU variable. Once this is done, the variable can be manipulated with the **SYSREAD** and **SYSWRITE** commands. The status of any variable can be determined with the **SYSTATUS\$** function at any time.

The forms of these statements are:

```
SYSREAD <variable name>,[NOWAIT],[frequency]
SYSWRITE <variable name>,[NOWAIT],[frequency]
<status string> = SYSTATUS$ ( <variable name> )
```

| Parameter | Description |
|--------------|---|
| VariableName | The name of the local MegaBasic variable. In the SYSREAD statement, this argument is the destination in the PCM for data to be read from the PLC CPU. In the SYSWRITE statement, it is the source for data to be written to the PLC CPU. Any type conversion specified in the SYSLINK command that associated the MegaBasic variable with the PLC data is performed. |
| NOWAIT | <p>When this option is used, the MegaBasic program continues to execute following the SYSREAD or SYSWRITE command, rather than wait for the CPU to respond with the data. The NOWAIT option is highly recommended for time-critical PCM applications.</p> <p>A MegaBasic logical interrupt is generated when the transfer has completed. This allows the program to handle backplane traffic asynchronously with MegaBasic program execution. (Refer to chapter 5, <i>Advanced MegaBasic Programming</i>, for more information on backplane interrupts.) The SYSTATUS\$ function can then be used to determine when the data has arrived and if there are any errors. This mode is useful when the CPU response time is long or when data is being transferred continuously.</p> <p>When NOWAIT is not specified (NORMAL or WAIT mode), the MegaBasic program waits until the CPU has transferred the data before continuing to execute the next line in the program. WAIT mode is the default.</p> |

| Parameter | Description |
|-----------|--|
| Frequency | <p>Both the SYSREAD and SYSWRITE statements may include this argument, which tells the PCM to transfer the data at a periodic rate. The frequency argument is an integer from 1 to 65,535, which specifies the number of milliseconds between each transfer. This argument may only be specified when the NOWAIT argument is also supplied. The frequency argument should not be less than the CPU sweep time.</p> <p>When the value of the frequency argument is zero, the variable is transferred as often as possible. The minimum time for a transfer is once per CPU sweep, assuming that all system communication processing by the CPU can be done in its allotted window and that the size of the variable is less than 256 bytes. Transfer times will be longer if the variable size is greater than 256 bytes or the CPU is heavily loaded with communication processing.</p> <p>If the number of milliseconds specified by the frequency argument is shorter than the time the variable can be transferred in, the variable is transferred as fast as possible. For fast update times, specify a frequency argument of zero. This results in the fastest possible update time and requires less internal processing by the PCM than using a small frequency argument, since no timers are associated with the transfers.</p> <p>For repeated transfers, the MegaBasic program is periodically interrupted in order to move data in and out of MegaBasic variables. If logical interrupts are disabled, either explicitly or by executing an instruction which takes a long time, the variables are no longer updated. For the SYSREAD statement, the PCM continues to read data from the PLC. When interrupts are re-enabled, the MegaBasic variable is updated to the most recent PLC value. For the SYSWRITE statement, writes to the PLC are suspended until interrupts are enabled.</p> <p>If the frequency argument is used with a SYSREAD or SYSWRITE statement to specify the repetitive update of a variable, and at some time the program wishes to stop repetitive transfers, the program must execute a new SYSREAD or SYSWRITE command for the variable without the frequency argument. The frequency of transfers can also be changed by executing a new SYSREAD or SYSWRITE command with a new frequency argument.</p> <p>When a Series 90-70 PCM MegaBasic application needs to obtain PLC CPU data as often as it is updated in the CPU (once per sweep), the PLC communications window mode should be set to RUN TO COMPLETION (default setting). If there are too many requests or if there are several PCMs, this may actually cause the PLC CPU watchdog timer to halt the CPU. In this case, reduce the number of requests or the number of PCMs in the system, or run in LIMITED WINDOW mode.</p> <p>The NOWAIT frequency should be greater than or equal to the PLC CPU sweep time when the PLC CPU communications window mode is LIMITED WINDOW.</p> |

Status Record

If a **SYSREAD** or **SYSWRITE** is called without the **NOWAIT** argument, any errors that occur during the transfer cause an error trap. Otherwise, the **SYSTATUS\$** function is used to monitor variables that have been **SYSLINK**ed to the PLC CPU. It is called with a MegaBasic variable name as its argument. It returns a string record containing three integers:

1. The first integer holds the current status of the PLC variable.
2. The second integer contains a status history.
3. The third integer specifies the time that the variable was last updated.

The status record is defined as follows:

```
Struct Integer CUR_STAT, integer STAT_HIST, integer STAT_TIME
```

Current Status

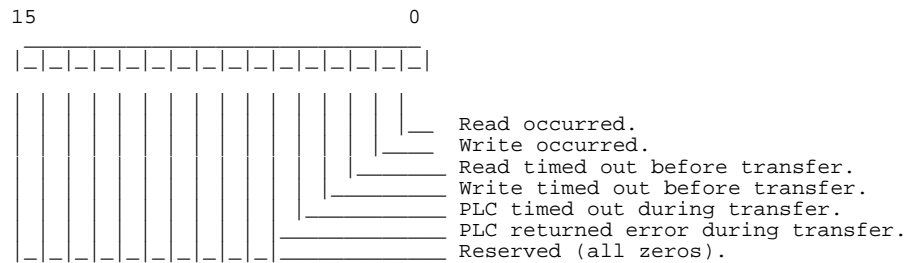
The possible valid **CUR_STAT** codes are listed as hexadecimal values in the following table:

| Code | Status | Description |
|------|----------------|---|
| 0001 | STABLE | Variable has been linked, but is not currently being transferred. |
| 0002 | READ_PENDING | Variable is being read from the CPU. |
| 0003 | READ_RECEIVED | Variable has been read and is waiting for a timer to start a new read. |
| 0004 | READ_TIMEOUT | Timer to start a new read has expired, but the previous read request has not completed. |
| 0005 | WRITE_PENDING | Variable is being written to the CPU. |
| 0006 | WRITE_RECEIVED | Variable has been written and is waiting for a timer to start a new write. |
| 0007 | WRITE_TIMEOUT | Timer to start a new write has expired, but the previous write request has not completed. |
| 0008 | WRITE_FINISHED | Variable is waiting to be refreshed with the latest MegaBasic value, so that the new contents can be written to the PLC. |
| 000B | NO_CPU | No response from the CPU. Status code 000B is an error code that occurs when the PLC does not respond to the PCM within 10 seconds of the PCM's request. This generally indicates the PLC CPU is not functioning. |
| 000C | XFER_REJECT | <p>Error response from the CPU. Status code 000C is an error code that occurs when an invalid transfer request passes the PCM's error checking but is stopped by the PLC. The most common reason for this is that a transfer to a PLC data area exceeded the bounds of that area.</p> <p>For example, if the PLC has 8K of register space, a transfer of any length to %R9000 would result in a status of 000C, as would a 4-byte transfer to %R8192.</p> |

Status History

The `STAT_HIST` contains information on what has happened to the variable since the last time the status was checked. The status history contains several bit flags, which are set upon various events. Consequently, it may be more convenient to deal with it as a bit string rather than an integer.

The status history is defined as follows:



Each time one of these events occurs, the corresponding bit in the status history is set. The bits are cleared each time the `SYSTATUS$` function is called.

The timeout bits may be used with repeated backplane transfers to verify that a repeated transfer is occurring at the designated rate. If either the PLC or PCM is delayed longer than the frequency value specified in a **SYSREAD** or **SYSWRITE** statement, the read or write timeout bit is set. This is particularly useful when sampling PLC values to ensure that the sampling is taking place at the designated rate.

Status Time

The **STAT_TIME** field of the status record may be used to find out the last time a variable was transferred. This time is the number of milliseconds since the beginning of the current day, counted from 0:00:00.000 (midnight).

Each read or write in a Series 90-70 PCM takes about 3 milliseconds of internal processing time on the PCM, plus 10 microseconds for each byte of the transfer. If the type of PLC variable is different from the MegaBasic variable type, additional time is required to convert each byte. Transfers in Series 90-30 PCMs are slower.

If several variables are transferred as often as possible, and the PLC sweep time is small, it is possible that the PCM may spend most of its time transferring data; the MegaBasic program will execute slowly. In this case, the response of the application may be improved by transferring the data less frequently.

For a sample MegaBasic program which uses `SYSLINK`, `SYSREAD`, and `SYSWRITE` extensively, see appendix E, *Example MegaBasic Program*.

UNLINK Statement

The **UNLINK** statement is used when a MegaBasic program no longer needs to transfer a variable that has been **SYSLINK**ed. The **UNLINK** statement tells the PCM that it is no longer necessary to maintain the data structure set up by the PCM to track the variable. This structure can then be reused to **SYSLINK** another variable.

The **UNLINK** statement has the form:

```
UNLINK <variable_name>
```

Once a variable has been **UNLINK**ed, it cannot be used with **SYSREAD**, **SYSWRITE**, or **SYSTATUS\$** until it is **SYSLINK**ed again. Any attempt to do so results in a “Remote variable unknown” error.

Data Coherency

When a frequency argument is used with **SYSREAD** or **SYSWRITE**, the variable is periodically updated. Special steps must be taken to ensure that the PCM’s copy of a PLC CPU variable is not updated while the MegaBasic program is manipulating it. This is especially important for arrays of data.

All data updates take place at the end of a MegaBasic statement. Once an update begins, the next statement does not execute until the update is completed. Consequently, coherency of a PLC variable is guaranteed if it can be processed in one statement.

Whenever a PLC variable is manipulated by more than one statement, you must ensure that an update of the variable between instructions does not cause erroneous results. To ensure that the variable is not corrupted, copy it to a temporary variable and use the copy for subsequent processing. Since the copy can be made with a simple assignment statement (one instruction), the temporary variable is guaranteed to contain a coherent copy of the PLC variable.

MegaBasic has two data types that enable large blocks of data to be copied with a single assignment statement. The first is the **vector** which can be used to copy arrays of numbers. The second is the **stringrecord**, which allows arbitrary groupings of data, defined within the same record, to be transferred.

After the PCM updates a variable with a **SYSREAD** or **SYSWRITE** command, either the PCM or CPU can change its copy of the variable. There is no guarantee that both copies of the variable are the same, unless you take steps to ensure that only one program updates the variable and always notifies the other about changes to the variable.

Accessing the PCM's LEDs

The PCM has three green LEDs, located at the top of the front panel. The topmost LED is designated as the BOARD OK LED. During normal operation, this LED is always on. The two LEDs below the BOARD OK LED are designated as USER1 and USER2. These LEDs may be configured to indicate serial port or backplane activity, or they may be manipulated directly by a user program. The default configuration is that USER1 flashes when there is activity on port 1; USER2 flashes when there is activity on port 2.

The configuration of the PCM LEDs cannot be changed with Logicmaster 90 software. However, it may be done with a **PCMEXEC.BAT** file if your PCM has firmware version 2.50 or greater. See appendix D, *PCM Batch Files*, for information on PCM batch files and appendix C, *PCM Commands*, for information on the **B (Configure LEDs)** command. The PCOP configuration editor can also be used to give control of the LEDs to a MegaBasic program (see GFK-0487).

The **SET_LED** statement is used by the MegaBasic program when an LED is configured to be under the control of MegaBasic. This utility can be used to set an LED's state to **ON**, **OFF**, **BLINK ONCE**, or **BLINK CONTINUOUSLY**.

The format of the **SET_LED** statement is:

```
Set_led led_number, operation_code
```

| Parameter | Description |
|----------------|---|
| LEDNumber | The number of the user-configurable LED, either 1 or 2. |
| Operation Code | The state of the LED. The four states are: 1 = Turn LED on. 2 = Turn LED off. 3 = Blink LED once. 4 = Blink LED continuously. |

If the LED is not configured to be under MegaBasic control, the **SET_LED** command has no effect. However, no error is returned. The operating system maintains virtual LEDs for each task, but only the configured task actually causes the physical LED state to change.

Section 3: MegaBasic Programming Examples

There are two examples in this section, one showing the steps to develop a simple MegaBasic program and the second showing how to get the example program **SAMPLE.PGM** running on the PCM. These examples assume the PCM has been configured using Logicmaster 90 for **BASIC** mode or by PCOP for BASIC operation.

Program development is begun by connecting the programming cable from the programmer to the PCM, setting the default directory to **\PCOP\EXAMPLES.PCM**, and typing **TERMF** to start up the TERMF terminal emulation program. Then, press the PCM Restart/Reset pushbutton for 10 seconds to place the PCM in **PROGRAM** mode. The MegaBasic banner is displayed on the screen, followed by the “Ready” prompt.

```
MegaBasic Version 5.602, under PCM VTOS v2.50
IEEE/Software floating point on an 80186/88 CPU

Copyright (C)1985-1990 by Christopher Cochran
MegaBasic Support BBS: 415-459-0896, PO Box 723
Fairfax, CA. USA 94930 -- Serial #0000

Ready
```

Now, enter the single line of the program shown below. After typing the line, save the program to the RAM Disk as **BASIC.PGM** using the **SAVE** command. Answer **Y** to the program creation prompt:

```
Ready
10 print "hello, world"
Ready
save basic.pgm
RAM:BASIC.PGM file not found, create it? y
      2 lines      20 code bytes
Ready
```

When a soft reset is initiated, the MegaBasic banner is displayed and the program is executed:

```
MegaBasic Version 5.602, under PCM VTOS v2.50
IEEE/Software floating point on an 80186/88 CPU

Copyright (C)1985-1990 by Christopher Cochran
MegaBasic Support BBS: 415-459-0896, PO Box 723
Fairfax, CA. USA 94930 -- Serial #0000

hello, world
```

After pressing the PCM Restart/Reset pushbutton again for 10 seconds to place the PCM in **PROGRAM** mode, the example program distributed with TERMF and PCOP can be transferred to the PCM. Because this is a large program, there is a long delay after executing the **LOAD** command before the PCM again displays the “Ready” prompt. Note that the **LOAD** command must specify PC: as the source since the program is being loaded from the programmer.

```
MegaBasic Version 5.602, under PCM VTOS v2.50
IEEE/Software floating point on an 80186/88 CPU

Copyright (C)1985-1990 by Christopher Cochran
MegaBasic Support BBS: 415-459-0896, PO Box 723
Fairfax, CA. USA 94930 -- Serial #0000

Ready
load pc:sample.pgm
405 lines      15,529 code bytes
Ready
```

Now save this program as **BASIC.PGM**. Since the old copy of BASIC.PGM still exists on the RAM Disk, the PCM prompts as to whether to overwrite the old copy.

```
Ready
save ram:basic.pgm
RAM:BASIC.PGM file already exists, OK? y
405 lines      15,529 code bytes
Ready
```

This example program accesses one of the utility packages, **VT100_5.PGM**, included with the PCM programming software. **VT100_5.PGM** contains routines for writing to a VT100 compatible screen. Since **VT100_5.PGM** must be available to the main program following a reset, it is loaded from the **EXAMPLES.PCM** directory into the PCM RAM Disk.

```
Ready
load pc:vt100_5.pgm
Into a new workspace? n
150 lines      3930 code bytes
Ready
save ram:
RAM:VT100_5.PGM file not found, create it? y
150 lines      3930 code bytes
Ready
```

The **DIR** command shows that the RAM Disk contains three files: **BASIC.PGM**, **VT100_5.PGM**, and **HARDEXEC.BAT**. **HARDEXEC.BAT** is the startup program used by the PCM after a hard reset.

```
Ready
dir
VT100_5.pgm  BASIC.pgm  HARDEXEC.bat
Ready
```

If a soft reset is initiated, the PCM comes up executing the example program. By following the displayed instructions, this program can be used to monitor and change references in the PLC's %R, %I, and %Q tables.

```

10 Rem          *** EXAMPLE MEGABASIC PROGRAM FOR PCM ***
20 Rem
30 Rem  This program implements a simple operator interface, which can
40 Rem  be easily modified for a particular application. This program
50 Rem  provides examples of the most commonly used MegaBasic commands
60 Rem  and extensions. It can be run using either port of the PCM and
70 Rem  assumes that a VT100, OIT or PC running Termf is connected.
80 Rem
90 Rem  First, access the MegaBasic subroutine package which handles
100 Rem screen formatting. It must have been previously loaded into RAM.
110 Rem
120 Access "RAM:VT100_5.PGM"

130 Rem  Now assign MegaBasic channel number #5 to whatever port is
140 Rem  is being used for the display. The screen formatting package
150 Rem  accessed above assumes that #5 is the output device
160 Rem
170 Open #5,"COM1:"

180 Rem  Disable control-C processing so that it doesn't interfere with
190 Rem  the keyboard handling routines. Since automatic control-C
200 Rem  detection will be disabled, the program must do its own
210 Rem  checking, if a control-C stop is desired.
220 Rem
230 Param(1)=1

240 Rem  Assign each menu a number, for switching between the menus.
250 Rem
260 Def integer MAIN_MENU
270 Def integer REG_MENU
280 Def integer INPUT_MENU
290 Def integer OUTPUT_MENU
300 MAIN_MENU = 1
310 REG_MENU = 2
320 INPUT_MENU = 3
330 OUTPUT_MENU = 4

340 Rem  All of the menus are implemented as subroutines and are called
350 Rem  from the main loop below. A variable called NEXT_MENU is used to
360 Rem  keep track of the next menu to display. When the user wants to
370 Rem  change menus, the active menu subroutine sets the NEXT_MENU
380 Rem  variable and returns to the main loop. To add a new menu, define
390 Rem  a new number for it, as above, write a subroutine to implement
400 Rem  the menu, and add another case statement in the loop below.
410 Rem
420 Def integer NEXT_MENU
430 NEXT_MENU = MAIN_MENU;      Rem Run the main menu first
440 Repeat
450   Case begin on NEXT_MENU
460     Case MAIN_MENU
470       DO_MAIN_MENU
480     Case REG_MENU
490       DO_REG_MENU
500     Case INPUT_MENU
510       DO_INPUT_MENU
520     Case OUTPUT_MENU
530       DO_OUTPUT_MENU
540   Case end
550 Next

```



```

560 Rem This is the main menu subroutine. It displays the various
570 Rem options and checks for input. Note that this is the only menu
580 Rem in which the user can type a control-C to exit. This is optional.
590 Rem
600 Def proc DO_MAIN_MENU
610 CLS; Rem Clear the screen
620 ATTR; Rem Reset line attributes
630 CUR 4,10; Rem Position cursor to row 4 column 10
640 Print #5, DW_DH_TOP$;
650 Print #5,"EXAMPLE MAIN MENU"; Rem Print top half of banner
660 CUR 5,10
670 Print #5, DW_DH_BOT$;
680 Print #5,"EXAMPLE MAIN MENU"; Rem Print bottom half of banner
690 CUR 9,25
700 Print #5,"1 - DISPLAY REGISTERS"; Rem Print the rest of the menu
710 CUR 11, 25
720 Print #5,"2 - DISPLAY INPUTS",
730 CUR 13, 25
740 Print #5,"3 - DISPLAY OUTPUTS",
750 CUR 15, 25
760 Print #5,"4 - EXIT PROGRAM",
770 CUR 21, 24
780 ATTR BLINK; Rem Make the next line blink
790 Print #5,"*** ENTER NUMBER ***",
800 ATTR
810 Rem The following loop continuously checks for user input
820 Rem
830 Repeat
840 KEY$=inchr$(5,1,"",0,0); Rem Check for a character
850 Case begin on KEY$
860 Case "1"; Rem Display register menu
870 NEXT_MENU = REG_MENU
880 Return
890 Case "2"; Rem Display input menu
900 NEXT_MENU = INPUT_MENU
910 Return
920 Case "3"; Rem Display output menu
930 NEXT_MENU = OUTPUT_MENU
940 Return
950 Case "4"; Rem Exit the program
960 Print
970 Stop
980 Case chr$(03); Rem Handle Control C
990 Print
1000 Stop
1010 Case end; Rem Ignore all other characters
1020 Next

1030 Rem This is the register display subroutine. It displays a block
1040 Rem of eight registers at a time. The user can change the value of
1050 Rem a register or display a different group of registers. No error
1060 Rem checking is done, and the display freezes while the operator
1070 Rem is entering data.
1080 Rem
1090 Def proc DO_REG_MENU

1100 Rem
1110 Local BASE_REG%; Rem Sets which registers to display
1120 Local WRITE_REG%; Rem Used for writing to PLC
1130 Local WRITE_DATA%; Rem Used for writing to PLC
1140 Local I%; Rem General purpose loop counter
1150 Local CMD_LINE$; Rem Used to write command line
1160 Local CLR_CMD_LINE$; Rem Used to clear command line
1170 Dim integer REG_ARRAY(7); Rem Used for reading PLC

```

```

1180 Rem
1190 Rem The following few lines draw the static part of the register
1200 Rem menu screen, using the same format as the main menu
1210 Rem
1220 CLS
1230 ATTR
1240 CUR 4,10
1250 Print #5, DW_DH_TOP$,
1260 Print #5,"REGISTER DISPLAY",
1270 CUR 5,10
1280 Print #5, DW_DH_BOT$,
1290 Print #5,"REGISTER DISPLAY",
1300 CUR 20,20
1310 Print #5,"1 - DISPLAY NEW REGISTERS",
1320 CUR 21, 20
1330 Print #5,"2 - WRITE A REGISTER",
1340 CUR 22, 20
1350 Print #5,"3 - EXIT TO MAIN MENU",
1360 CUR 24, 24
1370 ATTR BLINK
1380 Print #5,"*** ENTER NUMBER ***",
1390 ATTR
1400 Rem Build some strings for handling the command line
1410 Rem
1420 CMD_LINE$ = ATTR$(BLINK)+"*** ENTER NUMBER ***"+ATTR$(ATTR_OFF)
1430 CLR_CMD_LINE$ = CUR$(24,24)+(" " * 32)+CUR$(24,24)

1440 Rem SYSLINK the REG_ARRAY variable with %R1 in the PLC.
1450 Rem The SYSREAD command below will then copy the 8 registers
1460 Rem starting at %R1 to REG_ARRAY.
1470 Rem
1480 SYSLINK REG_ARRAY, "%R1", UINT
1490 BASE_REG% = 1

1500 Rem This is the main processing loop for the register menu
1510 Rem
1520 Repeat
1530     SYSREAD REG_ARRAY; Rem Read PLC registers and display them
1540     For I% = 0 to 7
1550         CUR 9+I%, 27
1560         Print #5, "%R ", %"5I5", BASE_REG%+I%," ",
1570         ATTR REVERSE
1580         Print #5, %"5I5", REG_ARRAY(I%),
1590         ATTR
1600     Next I%

1610     Rem Check for input and process if necessary. Note that the
1620     Rem Input commands used below will cause the display to freeze
1630     Rem while the operator is entering data. It is possible to handle
1640     Rem operator input using the inchr$ statement, or with NOWAIT_READS
1650     Rem of the keyboard, however, a fairly complicated state machine
1660     Rem would have to be used to figure out what to do when each key
1670     Rem is struck.
1680     Rem
1690     KEY$=inchr$(5,1,"",0,0)
1700     Case begin on KEY$
1710         Case "1"

```

```

1720          Rem To display new registers, first UNLINK with
1730          Rem the old registers and SYSLINK with the new ones.
1740          Rem The SYSREAD at the top of the loop will then
1750          Rem read the registers from the new location
1760          Rem
1770          Print #5, CLR_CMD_LINE$,
1780          Input #5, "NEW REGISTER TO DISPLAY ? ", BASE_REG%
1790          UNLINK REG_ARRAY
1800          SYSLINK REG_ARRAY, "%R"+str$(BASE_REG%,"5I5"), UINT
1810          Print #5, CLR_CMD_LINE$,
1820          Print #5, CMD_LINE$,
1830          Case "2"

1840          Rem To write a register, SYSLINK with the register
1850          Rem which is to be written and do a SYSWRITE to write
1860          Rem the data. Since the register to be written always
1870          Rem changes, UNLINK the register after writing it
1880          Rem
1890          Print #5, CLR_CMD_LINE$,
1900          Input #5, "REGISTER TO WRITE ? ", WRITE_REG%
1910          Print #5, CLR_CMD_LINE$,
1920          Input #5, "VALUE ? ", WRITE_DATA%
1930          SYSLINK WRITE_DATA%, "%R"+str$(WRITE_REG%,"5I5"), UINT
1940          SYSWRITE WRITE_DATA%
1950          UNLINK WRITE_DATA%
1960          Print #5, CLR_CMD_LINE$,
1970          Print #5, CMD_LINE$,
1980          Case "3";          Rem Go back to main menu
1990          NEXT_MENU = MAIN_MENU
2000          UNLINK REG_ARRAY
2010          Return
2020          Case end;          Rem Ignore all other characters
2030 Next

2040 Rem This is the input display subroutine. It displays a block
2050 Rem of sixty four inputs at a time. The user can change the value of
2060 Rem an input or display a different group of inputs. No error
2070 Rem checking is done, and the display freezes while the operator
2080 Rem is entering data.
2090 Rem
2100 Def proc DO_INPUT_MENU 2110 Rem
2120 Local BASE_INPUT%;          Rem Sets which inputs to display
2130 Local WRITE_INPUT%;          Rem Used for writing to PLC
2140 Local INPUT_DATA%;          Rem Used for writing to PLC
2150 Local I%;          Rem General purpose loop counter
2160 Local CMD_LINE$;          Rem Used to write command line
2170 Local CLR_CMD_LINE$;          Rem Used to clear command line
2180 Dim integer INPUT_ARRAY(7);          Rem Used for reading PLC
2190 Rem
2200 Rem The following few lines draw the static part of the input
2210 Rem menu screen, using the same format as the main menu
2220 Rem
2230 CLS
2240 ATTR
2250 CUR 4,9
2260 Print #5, DW_DH_TOP$,
2270 Print #5,"INPUT TABLE DISPLAY",
2280 CUR 5,9
2290 Print #5, DW_DH_BOT$,

```

```

2300 Print #5,"INPUT TABLE DISPLAY",
2310 CUR 20,20
2320 Print #5,"1 - DISPLAY NEW SET OF INPUTS",
2330 CUR 21, 20
2340 Print #5,"2 - WRITE AN INPUT",
2350 CUR 22, 20
2360 Print #5,"3 - EXIT TO MAIN MENU",
2370 CUR 24, 24
2380 ATTR BLINK
2390 Print #5,"*** ENTER NUMBER ***",
2400 ATTR
2410 Rem Build some strings for handling the command line
2420 Rem
2430 CMD_LINE$ = ATTR$(BLINK)+"*** ENTER NUMBER ***"+ATTR$(ATTR_OFF)
2440 CLR_CMD_LINE$ = CUR$(24,24)+(" "32)+CUR$(24,24)

2450 Rem SYSLINK the INPUT_ARRAY variable with %I1 in the PLC.
2460 Rem The SYSREAD command below will then copy the 64 inputs
2470 Rem starting at %I1 to INPUT_ARRAY.
2480 Rem
2490 SYSLINK INPUT_ARRAY, "%I1", BYTE
2500 BASE_INPUT% = 1

2510 Rem This is the main processing loop for the input menu
2520 Rem
2530 Repeat
2540   SYSREAD INPUT_ARRAY; Rem Read PLC inputs and display them
2550   For I% = 0 to 7
2560     CUR 9+I%, 26
2570     Print #5, "%I ", %"5I5", 8*I%+BASE_INPUT%, " ",
2580     ATTR REVERSE
2590     Print #5, %"8B8", INPUT_ARRAY(I%),
2600     ATTR
2610   Next I%

2620   Rem Check for input and process if necessary. Note that the
2630   Rem Input commands used below will cause the display to freeze
2640   Rem while the operator is entering data. It is possible to handle
2650   Rem operator input using the inchr$ statement, or with NOWAIT_READS
2660   Rem of the keyboard, however, a fairly complicated state machine
2670   Rem would have to be used to figure out what to do when each key
2680   Rem is struck.
2690   Rem
2700   KEY$=inchr$(5,1,"",0,0)
2710   Case begin on KEY$
2720     Case "1"

2730       Rem To display new inputs, first UNLINK with
2740       Rem the old inputs and SYSLINK with the new ones.
2750       Rem The SYSREAD at the top of the loop will then
2760       Rem read the inputs from the new location
2770       Rem
2780       Print #5, CLR_CMD_LINE$,
2790       Input #5, "NEW INPUT TO DISPLAY ? ", BASE_INPUT%
2800       UNLINK INPUT_ARRAY
2810       SYSLINK INPUT_ARRAY, "%I"+str$(BASE_INPUT%,"5I5"), BYTE
2820       Print #5, CLR_CMD_LINE$,
2830       Print #5, CMD_LINE$,
2840     Case "2"

```

```

2850          Rem To write an input, SYSLINK with the input
2860          Rem which is to be written and do a SYSWRITE to write
2870          Rem the data. Since the input to be written always
2880          Rem changes, UNLINK the input after writing it
2890          Rem
2900          Print #5, CLR_CMD_LINE$,
2910          Input #5, "INPUT TO WRITE ? ", WRITE_INPUT%
2920          Print #5, CLR_CMD_LINE$,
2930          Input #5, "VALUE ? ", INPUT_DATA%
2940          SYSLINK INPUT_DATA%, "%I"+str$(WRITE_INPUT%, "5I5"), BOOL
2950          SYSWRITE INPUT_DATA%
2960          UNLINK INPUT_DATA%
2970          Print #5, CLR_CMD_LINE$,
2980          Print #5, CMD_LINE$,
2990          Case "3";                      Rem Go back to main menu
3000          NEXT_MENU = MAIN_MENU
3010          UNLINK INPUT_ARRAY
3020          Return
3030      Case end;                          Rem Ignore all other characters
3040 Next

3050 Rem  This is the output display subroutine. It displays a block
3060 Rem  of sixty four outputs at a time. The user can change the value of
3070 Rem  an output or display a different group of outputs. No error
3080 Rem  checking is done, and the display freezes while the operator
3090 Rem  is entering data.
3100 Rem
3110 Def proc DO_OUTPUT_MENU

3120 Rem
3130 Local BASE_OUTPUT%;                      Rem Sets which inputs to display
3140 Local WRITE_OUTPUT%;                    Rem Used for writing to PLC
3150 Local OUTPUT_DATA%;                    Rem Used for writing to PLC
3160 Local I%;                              Rem General purpose loop counter
3170 Local CMD_LINE$;                        Rem Used to write command line
3180 Local CLR_CMD_LINE$;                    Rem Used to clear command line
3190 Dim integer OUTPUT_ARRAY(7);           Rem Used for reading PLC

3200 Rem
3210 Rem The following few lines draw the static part of the input
3220 Rem menu screen, using the same format as the main menu
3230 Rem
3240 CLS
3250 ATTR
3260 CUR 4,8
3270 Print #5, DW_DH_TOP$,
3280 Print #5, "OUTPUT TABLE DISPLAY",
3290 CUR 5,10
3300 Print #5, DW_DH_BOT$,
3310 Print #5, "OUTPUT TABLE DISPLAY",
3320 CUR 20,20
3330 Print #5, "1 - DISPLAY NEW SET OF OUTPUTS",
3340 CUR 21, 20
3350 Print #5, "2 - WRITE AN OUTPUT",
3360 CUR 22, 20
3370 Print #5, "3 - EXIT TO MAIN MENU",
3380 CUR 24, 24
3390 ATTR BLINK
3400 Print #5, "***  ENTER NUMBER  ***",
3410 ATTR
3420 Rem Build some strings for handling the command line
3430 Rem
3440 CMD_LINE$ = ATTR$(BLINK)+"***  ENTER NUMBER  ***"+ATTR$(ATTR_OFF)
3450 CLR_CMD_LINE$ = CUR$(24,24)+(" "32)+CUR$(24,24)

```

```

3460 Rem SYSLINK the OUTPUT_ARRAY variable with %Q1 in the PLC.
3470 Rem The SYSREAD command below will then copy the 64 outputs
3480 Rem starting at %Q1 to OUTPUT_ARRAY.
3490 Rem
3500 SYSLINK OUTPUT_ARRAY, "%Q1", BYTE
3510 BASE_OUTPUT% = 1

3520 Rem This is the main processing loop
3530 Rem
3540 Repeat
3550   SYSREAD OUTPUT_ARRAY; Rem Read PLC outputs and display them
3560   For I% = 0 to 7
3570     CUR 9+I%, 26
3580     Print #5, "%Q ", %"5I5", 8*I%+BASE_OUTPUT%," ",
3590     ATTR REVERSE
3600     Print #5, %"8B8", OUTPUT_ARRAY(I%),
3610     ATTR
3620   Next I%

3630   Rem Check for input and process if necessary. Note that the
3640   Rem Input commands used below will cause the display to freeze
3650   Rem while the operator is entering data. It is possible to handle
3660   Rem operator input using the inchr$ statement, or with NOWAIT_READs
3670   Rem of the keyboard, however, a fairly complicated state machine
3680   Rem would have to be used to figure out what to do when each key
3690   Rem is struck.
3700   Rem
3710   KEY$=inchr$(5,1,"",0,0)
3720   Case begin on KEY$
3730     Case "1"

3740       Rem To display new outputs, first UNLINK with
3750       Rem the old outputs and SYSLINK with the new ones.
3760       Rem The SYSREAD at the top of the loop will then
3770       Rem read the outputs from the new location
3780       Rem
3790       Print #5, CLR_CMD_LINE$,
3800       Input #5, "NEW OUTPUT TO DISPLAY ? ", BASE_OUTPUT%
3810       UNLINK OUTPUT_ARRAY
3820       SYSLINK OUTPUT_ARRAY, "%Q"+str$(BASE_OUTPUT%,"5I5"), BYTE
3830       Print #5, CLR_CMD_LINE$,
3840       Print #5, CMD_LINE$,
3850     Case "2"

3860       Rem To write an output, SYSLINK with the output
3870       Rem which is to be written and do a SYSWRITE to write
3880       Rem the data. Since the output to be written always
3890       Rem changes, UNLINK the output after writing it
3900       Rem
3910       Print #5, CLR_CMD_LINE$,
3920       Input #5, "OUTPUT TO WRITE ? ", WRITE_OUTPUT%
3930       Print #5, CLR_CMD_LINE$,
3940       Input #5, "VALUE ? ", OUTPUT_DATA%
3950       SYSLINK OUTPUT_DATA%, "%Q"+str$(WRITE_OUTPUT%,"5I5"), BOOL
3960       SYSWRITE OUTPUT_DATA%
3970       UNLINK OUTPUT_DATA%
3980       Print #5, CLR_CMD_LINE$,
3990       Print #5, CMD_LINE$,
4000     Case "3";           Rem Go back to main menu
4010       NEXT_MENU = MAIN_MENU
4020       UNLINK OUTPUT_ARRAY
4030       Return
4040   Case end;           Rem Ignore all other characters
4050 Next

```

Chapter 5

Advanced MegaBasic Programming

MegaBasic is a powerful implementation of the BASIC language which runs under twelve different operating systems and a host of different hardware configurations. One of the strengths of MegaBasic is that the language can be “extended” to support the underlying hardware. This chapter describes the additions, called **extensions** that GE Fanuc has made to MegaBasic. These additions or extensions allow MegaBasic to take full advantage of the special capabilities of the PCM and the Series 90 system.

This chapter contains the following sections:

| Section | Title | Description | Page |
|---------|---|--|------|
| 1 | MegaBasicError Codes | Section 1 lists common error codes from the PCM extensions to MegaBasic. | 5-3 |
| 2 | Screen Formatting Commands | Section 2 describes predefined control sequences for manipulating a VT100 compatible display. These functions and procedures are located in VT100.PGM | 5-5 |
| 3 | Accessing %P, %L, and Password-Protected Data | Section 3 describes definitions and procedures used to access user references not directly supported by the PCM backplane driver and to access data protected by a user password. These procedures are found in GENERIC.PGM | 5-16 |
| 4 | Access to PLC Fault Tables and PLC Status | Section 4 describes how to access the PLC and I/O fault tables from a MegaBasic program, using RD_FLT.PGM | 5-21 |
| 5 | Gathering PLC Information from MegaBasic Programs | Section 5 describes how to use UTILITY.PGM to obtain a variety of PLC CPU data. | 5-34 |
| 6 | Loading and Storing PCM Data Files Using TERMF | Section 6 describes how to load MegaBasic data files and utility packages in binary form to the PCM using TERMF and the PCM command interpreter. | 5-50 |
| 7 | Serial Port Setup with IOCTL and PORT_CTL.BIN | Section 7 describes how to change the PCM serial port configuration, send and detect serial breaks, and use the modem control and status signals of the serial ports in MegaBasic programs. | 5-52 |
| 8 | Timers and Logical Interrupts | Section 8 describes timer and backplane interrupts. | 5-57 |
| 9 | COMMREQs and Other Backplane Messages | Section 9 describes the processing of messages from COMMREQ function blocks and other sources. | 5-64 |

| Section | Title | Description | Page |
|---------|---|--|-------|
| 10 | AsynchronousSerial Input and Output | Section 10 describes the the NOWAIT I/O commands that support asynchronous serial communications. These commands include: <ul style="list-style-type: none">• NOWAIT_OPEN• NOWAIT_CLOSE• NOWAIT_READ• NOWAIT_WRITE• NOWAIT_SEEK• NOWAIT_READ_ABORT• NOWAIT_WRITE_ABORT | 5-91 |
| 11 | VME Functions | Section 11 describes the use of Series 90-70 VME functions as alternatives for communicating with a Series 90-70 PCM. | 5-100 |
| 12 | PCM Programming Example using MegaBasic | Section 12 provides a complete programming example containing PLC VME functions and EXAM and FILL MegaBasic statements. | 5-110 |
| 13 | Optimizing Backplane Communication | Section 13 describes how to maximize data throughput between the PCM and the PLC CPU. | 5-114 |

Section 1: MegaBasic Error Codes

The table in this section explains error codes returned by PCM extensions to MegaBasic.

Table 5-1. MegaBasic Error Codes

| Code | Status | Description/Corrective Action |
|------|---------------------------|--|
| 100 | MissingArgument | One or more arguments to a PCM extension is missing. |
| 101 | Argument Out of Range | One or more arguments to a PCM extension is out of bounds. |
| 102 | Argument of Wrong Type | One or more arguments to a PCM extension is of an invalid type. |
| 103 | InsufficientMemory | The PCM does not have enough system memory to complete the requested operation. This error is not to be confused with error 0, which occurs if the MegaBasic interpreter does not have enough memory. If this error occurs, configure MegaBasic to use less memory, leaving more for PCM system memory. |
| 104 | Remote Variable Unknown | The PCM could not complete a SYSLINK . This could be due to an invalid CPU name specified by SYSLINK , or the CPU may not be responding. |
| 105 | Too Many Remote Variables | Too many SYSLINKS are active. UNLINK those that are not used all of the time. |
| 106 | PCM Hardware Not Present | This error is generated if an MS-DOS version of MegaBasic tries to access the PCM extensions, or if the version number of an extension package does not match the PCM MegaBasic version number. |
| 107 | Not Enough Timers | The PCM has run out of system timers. Use fewer timers in the MegaBasic program. |
| 108 | Bad Timer Definition | An invalid argument has been passed to the TIMER statement. |
| 109 | Task Not Active | The MegaBasic program attempted to manipulate an LED that was not configured to be under the control of a user program. If your PCM has firmware version 2.50 or greater, you can use a PCMEEXEC.BAT file containing the B (Configure LEDs) command to assign the LED to your MegaBasic program. With lower firmware versions, you <u>must</u> use PCOP to configure the LED. |
| 110 | Bad LED Definition | Either an invalid LED number or invalid mode was specified by the MegaBasic program. |
| 111 | Duplicate Remote Variable | A MegaBasic local variable being SYSLINKed is the object of another SYSLINK call. A MegaBasic local variable can be SYSLINKed to only one PLC CPU variable. |

Table 5-1. MegaBasic Error Codes (cont'd)

| Code | Status | Description/Corrective Action |
|------|-------------------------------|--|
| 112 | Backplane Transfer Failure | A backplane transfer cannot be completed. This could be caused by a timeout waiting for a response from the PLC CPU, attempting to SYSREAD or SYSWRITE more data than a particular PLC memory type contains, or attempting to write into a protected region of PLC CPU memory. |
| 113 | Undefined Local Variable | This error occurs if a MegaBasic local variable is made the object of a SYSLINK call before it has been declared. |
| 114 | Improper Variable Type | This error occurs if a string array or an array with more than three dimensions is SYSLINKed . |
| 115 | CPU Name String Too Long | CPU name strings must be eight characters or less, including the ASCII NUL termination character. |
| 116 | Basic Extensions Incompatible | The internal revision numbers for the PCM extension packages do not match those of the interpreter. |
| 117 | Illegal Backplane Operation | A MegaBasic program tried to SYSWRITE a variable while it is in the process of being read, or a SYSREAD occurred while the variable is being written. |
| 118 | Backplane Timeout | The CPU backplane driver timed out during an attempted transfer. It is a good idea to check for this error on the first SYSLINK in a program, in case the CPU has not yet completed its initialization. |
| 119 | Illegal NOWAIT I/O Operation | A NOWAIT_READ or NOWAIT_WRITE is attempted after a channel has been closed. |
| 120 | Buffer Space Exceeded | The user program exceeded the maximum designated outstanding buffer space for NOWAIT I/O . The program must specify that this condition should cause an error; otherwise no error is posted. |
| 122 | Invalid IOCTL String | Either an invalid serial port setup string was passed to the IOCTL statement or the IOCTL\$() function, or MegaBasic attempted to reconfigure a serial port used by another task. |
| 123 | Device Unavailable | An invalid device name was passed to the OPEN statement or to the DIR or DESTROY command. |
| 124 | Serial Port Parity | A parity error was detected in received serial data. |
| 125 | Serial Port Overrun | An overrun error was detected in received serial data. |
| 126 | Serial Port Framing | A framing error was detected in received serial data. |
| 127 | Serial Port Multiple | Two or more different parity, overrun or framing errors were detected in received serial data. |

Section 2: Screen Formatting Commands

Because the PCM is often used with alphanumeric display devices, additional MegaBasic statements have been provided to deal with these devices. At present, the display statements are only available for terminals that use DEC VT100 compatible control sequences.

VT100.PGM supplies several procedures and functions, as well as predefined control sequences for driving VT100 style displays. A companion file, **VT100_5.PGM**, is also available. The difference between these two files is that **VT100.PGM** prints to STDOUT, while **VT100_5.PGM** prints to the device opened as #5.

To use the functions, procedures, and data structure definitions supplied in **VT100.PGM**, add the line:

```
xxx Access "vt100.pgm"
```

to the beginning of a MegaBasic program or in the PROLOGUE section of a user written package. The file **VT100.PGM** should have been loaded to the PCM RAM Disk before running a MegaBasic program which accesses it. The file is located in the \PCOP or \PCOP\EXAMPLES.PCM directory of your PC hard disk, depending on which release of PCOP or TERMF you use.

The functions and procedures found in **VT100.PGM** are summarized in the following tables and are fully described on the pages following these tables.

Table 5-2. VT100.PGM Functions and Procedures

| Function/ Procedure | Description |
|------------------------|--|
| CLS | A statement that prints the control sequence to clear the screen (<code>ERASE_SCREEN\$</code>). |
| CUR\$ ¹ | A function that takes row and column integer parameters and returns a control string that, when printed to the VT100 device, positions the cursor at the desired location. This function may be used to build up larger control and text sequences (screens) for later display. |
| CUR ¹ | A statement that takes row and column integer parameters, and prints the resulting cursor positioning string. |
| ATTR\$ ² | A function that takes a list of desired screen display attributes and returns a formatted Set Graphic Rendition (SGR) control string that, when printed to the VT100 device, sets the desired screen attributes to be used for the next text printed. The attributes defined by this package for use with the <code>ATTR\$</code> function and <code>ATTR</code> statement are <code>ATTRIB_OFF</code> , <code>BLINK</code> , <code>BOLD</code> , <code>UNDERSCORE</code> , and <code>REVERSE</code> . In addition, if the device supports additional SGR controls such as TERMF's controls for foreground and background color (parameter values 30 - 37 and 40 - 47), these controls may also be supplied to the <code>ATTR\$</code> function and the <code>ATTR</code> statement. |
| ATTR ² | A statement that takes a list of the desired screen display attributes and prints the resulting SGR control string to the VT100 device. See <code>ATTR\$</code> function above. |
| MV_CUR\$ ³ | A function that takes direction and count parameters, and returns the appropriate relative move cursor control string for use in relative cursor movement commands that may be printed at a later time (i.e., a relocatable section of a screen display). The direction is one of <code>C_UP</code> , <code>C_DN</code> , <code>C_RT</code> or <code>C_LE</code> , which are the cursor up, down, right and left direction constants. |
| MV_CUR ³ | A statement that takes direction and count parameters, and prints the appropriate relative cursor movement control string. See <code>MV_CUR\$</code> function above. |

¹ The row and column parameters are optional. The default row is the top row (row 1), and the default column is the leftmost column (column 1).

² The list of parameters is completely optional. The default SGR string, produced when no parameters are supplied, results in turning all screen attributes off.

³ The direction and count parameters are optional. The default direction is `C_UP`, and the default count is one location.

Several integer and string constants are also defined in the `VT100.PGM` package. These constants are summarized in the following table.

Table 5-3. Integer and String Constants

| Constant | Description |
|----------------|---|
| ATTRIB_OFF | An integer constant (= 0) used in the ATTR\$ function or ATTR statement parameter list to indicate that all existing screen display attributes are to be reset. Typically this is the first parameter in a sequence that also sets one or more attributes. When simply turning off all attributes, it is recommended that you use the ATTR\$ function or ATTR statement with no parameters. |
| BLINK | An integer constant (= 5) that turns on the blinking text screen display attribute. |
| BOLD | An integer constant (= 1) that turns on the high intensity/bold text screen display attribute. |
| UNDERSCORE | An integer constant (= 4) that turns on the underline text screen display attribute. |
| REVERSE | An integer constant (= 7) that turns on the foreground/background reversed text screen display attribute. |
| C_UP | An integer constant (= 1) used in the MV_CUR\$ function or MV_CUR statement to move the cursor up. |
| C_DN | An integer constant (= 2) used in the MV_CUR\$ function or MV_CUR statement to move the cursor down. |
| C_RT | An integer constant (= 3) used in the MV_CUR\$ function or MV_CUR statement to move the cursor right. |
| C_LF | An integer constant (= 4) used in the MV_CUR\$ function or MV_CUR statement to move the cursor left. |
| CPS | A string constant (= <esc>I) used as the prefix for many of the VT100 control strings. |
| CURHOMES | A string constant (= <esc>H) used to set the cursor position to the home position (row 1/column 1). |
| SAVECUR\$ | A string constant (= <esc>7) used to save the present cursor position and screen display attributes. |
| RESTORECUR\$ | A string constant (= <esc>8) used to restore the previously saved cursor position and screen display attributes. |
| DW_DH_TOP\$ | A string constant (= <esc>3) used to set the current line to double wide/double high top half display format. |
| DW_DH_BOT\$ | A string constant (= <esc>4) used to set the current line to double wide/double high bottom half display format. |
| SW_SH\$ | A string constant (= <esc>5) used to set the current line to single wide/single high display format. |
| DW_SH\$ | A string constant (= <esc>6) used to set the current line to double wide/single high display format. |
| ERASE_EOL\$ | A string constant (= <esc>K) used to erase the present line from the cursor to the end of the line. |
| ERASE_BOL\$ | A string constant (= <esc>1K) used to erase the present line from the beginning of the line to the cursor. |
| ERASE_LINES\$ | A string constant (= <esc>2K) used to erase the entire present line. |
| ERASE_BOT\$ | A string constant (= <esc>J) used to erase the screen from the cursor to the bottom of the screen. |
| ERASE_TOP\$ | A string constant (= <esc>1J) used to erase the screen from the top of the screen to the cursor. |
| ERASE_SCREEN\$ | A string constant (= <esc>2J) used to erase the entire screen. |

CLS

The **CLS** statement clears the VT100 display screen and sets the cursor at the home position (row 1, column 1).

The format for the **CLS** statement is:

xxx CLS

The following example uses the **CLS** statement:

```
50 Access "vt100.pgm"  
100 CLS  
110 Print "*" <— This is home position for the cursor"
```

CUR\$

The **CUR\$** function formats a cursor positioning string for use in constructing displays for the VT100 screen. The first integer parameter indicates the desired row; the second integer parameter indicates the desired column where the cursor is to be placed. Both parameters are optional, with the defaults being row 1 and column 1.

The format for the **CUR\$** function is:

xxx A\$ = CUR\$(row,col)

| Argument | Description |
|----------|--|
| Row | An integer number specifying the desired row. |
| Column | An integer number specifying the desired column where the cursor is to be located. |

In the following example, the string **A\$** is constructed so that the screen is erased. Then, a double high and wide “HELLO” is displayed in the center of the VT100 screen. Note that, when the VT100 line is in double wide mode, the cursor positioning is based on a 40-column line.

```

50 Access "vt100.pgm"
100 A$ = ERASE_SCREEN$ + CUR$(10, 20) + DW_DH_TOP$ + "HELLO"
101 A$(:0) := CUR$(11, 20) + DW_DH_BOT$ + "HELLO"
200 Print A$

```

In the next example, a column of numbers is printed down the left side of the screen on every third line, and then the cursor is placed at the home position.

```

50 Access "vt100.pgm"
100 For I% = 2 to 20 by 3
110   Print CUR$(I%), "%2i", I%,
120 Next
130 Print CUR$,

```

CUR

The **CUR** statement prints a cursor positioning control string to the VT100.

The format of the **CUR** statement is:

xxx CUR row, col

| Argument | Description |
|----------|---|
| Row | An integer number specifying the desired row. |
| Column | An integer number specifying the desired column where the cursor is to located. |

Internally, the **CUR** statement calls the **CUR\$** function with the parameters that were passed to the **CUR** statement and then prints the resulting string. As with **CUR\$**, both parameters are optional and default to the home position values (row 1, column 1).

In the following example, the screen is cleared, the cursor is positioned to the center of the screen, and then a hello message is printed.

```
50 Access "vt100.pgm"
100 CLS
110 CUR 12,35
120 Print "Hello Y'all"
```

In the next example, the operation that was performed in the second example for the **CUR\$** function is repeated using the **CUR** statement.

```
50 Access "vt100.pgm"
100 For I% = 2 to 20 by 3
110     CUR I%
120     Print %"2i",I%,
130 Next
140 CUR
```


ATTR\$

The **ATTR\$** function formats a Set Graphic Rendition (SGR) control string for later display on a VT100 terminal or other compatible display device. The optional list of parameters specifies the desired attributes for the following text to be displayed on the VT100 device. The list of attributes that may be specified includes the constants **ATTRIB_OFF**, **BLINK**, **BOLD**, **UNDERSCORE**, and **REVERSE**. When all parameters are omitted, the resulting control string turns all attributes off when printed to the VT100 device.

The format of the **ATTR\$** function is:

xxx A\$ = ATTR\$(p1, p2, ...)

| Argument | Description |
|----------|--|
| P1 | An integer number specifying the desired attribute number 1. |
| P2 | The desired attribute number 2. |

In the following example a string, consisting of an erase screen control string and various combinations of attributes and related text strings, is constructed and then printed to the VT100 device. The effect of the various attributes is additive until a clear screen or an **ATTRIB_OFF** is issued. The **ATTR\$** function can accept more than one attribute per call, and more than one attribute may be turned on in the string returned by the **ATTR\$** function.

```

50 Access "vt100.pgm"
60 Dim A$(300), CRLF$(2)
70 CRLF$ = chrseq$(13, 10)
100 A$ = ERASE_SCREEN$ + ATTR$(BLINK) + "Blink " + CRLF$
110 A$(0) := ATTR$(BOLD) + "Bold Blink " + CRLF$
120 A$(0) := ATTR$(UNDERSCORE) + "Bold Blink Underscore  "
130 A$(0) := CRLF$ + ATTR$(REVERSE)
140 A$(0) := "Bold Blink Underscore Reverse  " + CRLF$
150 A$(0) := ATTR$(ATTRIB_OFF, UNDERSCORE, REVERSE)
160 A$(0) := "Underscore Reverse  " + CRLF$
170 A$(0) := ATTR$(BOLD) + "Bold Underscore Reverse  " + CRLF$
200 Print A$

```

In the next example, a string, consisting of an SGR control string activating blink and bold modes, the text "Working" and finally an SGR control string for all attributes off, is constructed. The string **A\$** is then printed at the home position, along with text to show that the attributes had been turned off.

```

50 Access "vt100.pgm"
100 A$ = ATTR$(ATTRIB_OFF, BLINK, BOLD) + "Working" + ATTR$
200 Print CURHOME$, A$, " I think"

```

Note

The TERMF terminal emulator supports the use of color selection control strings, using parameter values 30 - 37 and 40 - 47 to select the foreground and background color.

ATTR

The **ATTR** statement formats and prints an SGR control string to a VT100 terminal, or other compatible display device. The optional list of parameters specifies the desired attributes for the following text to be displayed on the VT100 device. The list of attributes that may be specified includes the constants **ATTRIB_OFF**, **BLINK**, **BOLD**, **UNDERSCORE**, and **REVERSE**. When all parameters are omitted, the resulting control string turns all attributes off on the VT100 device.

The format of the **ATTR** statement is:

```
xxx ATTR p1, p2, ...
```

| Argument | Description |
|----------|--|
| P1 | An integer number specifying the desired attribute number 1. |
| P2 | The desired attribute number 2. |

In the following example, the VT100 screen is cleared, the attributes set to bold and underscore, some text is printed at the home position, and then the attributes are turned off.

```
50 Access "vt100.pgm"
100 CLS
110 ATTR BOLD, UNDERSCORE
120 Print "GE Fanuc Automation N.A."
130 ATTR
```

In the next example, the operation that was performed in the second example for the **ATTR\$** function is repeated using the **ATTR** statement.

```
50 Access "vt100.pgm"
100 Print CURHOME$
110 ATTR ATTRIB_OFF, BLINK, BOLD
120 Print "Working"
130 ATTR
140 Print " I think"
```

MV_CUR\$

The **MV_CUR\$** function formats a relative cursor movement control string for later display on a VT100 terminal, or other compatible display device. The optional list of parameters specifies the desired direction for the move and the count of how many cursor positions to move in the selected direction. The list of directions that may be specified are **C_UP**, **C_DN**, **C_RT**, and **C_LF**.

All parameters are optional. The count defaults to one. The direction defaults to **C_UP** for the first pair, and no action for the rest of the pairs. If the final parameter in the list is a direction specifier, the count associated with that move is one location.

The format of the **MV_CUR\$** function is:

```
xxx A$ = MV_CUR$(dir1, cnt1, dir2, cnt2, ...)
```

| Argument | Description |
|-------------------------------|--|
| Direction x and Count x | <p>Pairs of integers, specifying the direction and count for each cursor move. The dir1/cnt1 pair is processed first; then, the remaining pairs are added to the resulting control string before the function completes.</p> <p>In practical terms, the maximum number of parameters for this function is 4 (2 pairs), since any location on the VT100 screen can be reached in two relative moves of the cursor. The direction parameters have a range of 1 to 4 (C_UP to C_LF). The count parameters specify how many positions to move the cursor (default = 1) in the specified direction.</p> |

In the following example, a relocatable graphic string (**BIG_PLUS\$**) is created and then displayed at several locations on the VT100 screen. The only characters on the screen that are overwritten are those characters that are directly overlapped by the printable characters (“-” “|” “+” <sp>) of the **BIG_PLUS\$** graphic.

```

50 Access "vt100.pgm"
100 BIG_PLUS$ = MV_CUR$ + "|" + MV_CUR$(C_LF, 4, C_DN)
110 BIG_PLUS$( :0 ) := "-" + "-" + MV_CUR$(C_LF, 4, C_DN) + "|"
200 Print CUR$(10, 20), BIG_PLUS$
210 Print CUR$(20, 40), BIG_PLUS$

```

In the next example, the **MV_CUR\$** function is used to construct some line-drawing primitives (lines 70-85). Then, a procedure is defined to use these line drawing primitives (lines 100-190). Finally, the main program clears the screen, draws a few lines, and then puts the cursor toward the bottom of the screen at the end of the program.

```

50 Access "vt100.pgm"
70 LINE_UP$ = MV_CUR$(C_LF, 1, C_UP) +  "^"
75 LINE_DN$ = MV_CUR$(C_LF, 1, C_DN) +  "v"
80 LINE_RT$ =  ">"
85 LINE_LF$ = MV_CUR$(C_LF, 2) +  "<"
100 Def proc LINE_DRAW DIR%=C_UP, CNT%=1
105 Local I%, LINE_OUT$
110 Print "+",
115 Case begin on DIR%
120 Case C_UP
125   LINE_OUT$ = LINE_UP$
130 Case C_DN
135   LINE_OUT$ = LINE_DN$
140 Case C_RT
145   LINE_OUT$ = LINE_RT$
150 Case C_LF
155   LINE_OUT$ = LINE_LF$
160 Case end
165 For I% = 1 to CNT%
170   Print LINE_OUT$,
175 Next
180 Print MV_CUR$(C_LF),
185 Return
190 Proc end
1000 CLS
1010 LINE_DRAW C_RT, 20
1020 LINE_DRAW C_DN, 10
1030 LINE_DRAW C_LF, 10
1040 LINE_DRAW C_RT, 5
1050 LINE_DRAW C_UP, 5
1060 LINE_DRAW C_RT, 30
1070 CUR 23

```

MV_CUR

The **MV_CUR** statement formats and prints relative cursor movement control strings on a VT100 terminal, or other compatible display device. The optional list of parameters specifies the desired direction for each move and the count of how many cursor positions to move in the selected direction. The directions that may be specified are **C_UP**, **C_DN**, **C_RT**, and **C_LF**. All parameters are optional. The count defaults to one location to move. The direction defaults to **C_UP** for the first move, and no operation for subsequent moves.

The format of the **MV_CUR** statement is:

```
xxx MV_CUR dir1, cnt1, dir2, cnt2, ...
```

| Argument | Description |
|-------------------------------|---|
| Direction x and Count x | <p>Pairs of integers specifying the direction and count for each cursor move. The dir1/cnt1 pair is processed first; then, the remaining pairs, if any, are processed before the statement completes.</p> <p>In practical terms, the maximum number of parameters for this statement is 4 (2 pairs), since any location on the VT100 screen can be reached in two relative moves of the cursor. The direction parameters have a range of 1 to 4 (C_UP to C_LF). The count parameters specify how many locations to move the cursor (default = 1 location) in the specified direction.</p> |

In the following example, a procedure is defined (**BIG_PLUS**) that duplicates the action shown previously in the first example for the **MV_CUR\$** function.

```

50 Access "vt100.pgm"
100 Def proc BIG_PLUS
110 MV_CUR
120 Print "|",
130 MV_CUR C_LF, 4, C_DN
140 Print "- + -",
150 MV_CUR C_LF, 4, C_DN
160 Print "|"
170 Return
180 Proc end
1000 CUR 10, 20
1010 BIG_PLUS
1020 CUR 20, 40
1030 BIG_PLUS

```

In the next example, a simple procedure is defined and used to waste time by bouncing the cursor left and right.

```

50 Access "vt100.pgm"
100 Def proc WASTE_TIME
110 Local I%
120 For I% = 20 to 1 by -4
130   MV_CUR C_LF, I%, C_RT, I% - 1, C_LF, I% - 2, C_RT, I% - 3
140 Next
150 Return
160 Proc end
1000 CUR 10, 40
1010 WASTE_TIME

```

Section 3: Accessing %P, %L, and Password-Protected Data

GENERIC.PGM supplies definitions and procedures to access Series 90-70 PLC reference data types not directly supported by the PCM backplane driver (e.g., %P and %L) and to access data protected by a user password in the Series 90-30 or Series 90-70 PLCs. Attempting to **SYSWRITE** PLC data when PLC privilege level 2 is protected by a password will result in an error unless the PCM can gain access to level 2, as described here.

In addition, **GEN_TEST.PGM** is a sample program using **GENERIC.PGM**. **GENERIC.DOC** is a line-number referenced documentation of **GENERIC.PGM** and **GEN_TEST.PGM**. These files are installed with TERMF in the \PCOP or \PCOP\EXAMPLES.PCM directory of your PC hard disk, depending on which release of PCOP or TERMF you use.

To use the functions in **GENERIC.PGM**, add the line:

```
xxx Access "RAM:generic.pgm"
```

at the beginning of a MegaBasic program. The file **GENERIC.PGM** should be loaded to the PCM RAM Disk before running a MegaBasic program which accesses it.

The procedures found in **GENERIC.PGM** are listed in the following table:

| Procedure | Description |
|------------|--|
| CHG_PRIV | Change the current privilege level for MegaBasic communication with the PLC. |
| SMSG_WTEXT | Format and send a message to the PLC. |

Several shared structures and constants are also defined in **GENERIC.PGM**. These include:

| Constant/Structure | Description |
|--------------------|--|
| SMSG_TEXT\$ | The string used with SMSG_WTEXT which contains the program and block names. |
| ACC_CODE\$ | A string containing the access codes for various PLC reference data types. |
| R_TM\$ | Definition of request code to read task memory (%P). |
| R_PBM\$ | Definition of request code to read program subblock memory (%L). |
| W_TM\$ | Definition of request code to write task memory (%P). |
| W_PBM\$ | Definition of request code to write program sub-block memory (%L). |

CHG_PRIV

CHG_PRIV is used to change the privilege level for communication between the PCM MegaBasic task and the PLC CPU Service Request Processor (SRP). For more information on privilege levels and passwords, refer to the *Series 90-70 Programmable Controller Installation Manual*, GFK-0262, and the *Logicmaster 90-70 Programming Software User's Manual*, GFK-0263.

The format for using the **CHG_PRIV** procedure is:

```
xxx CHG_PRIV n, str
```

| Parameter | Description |
|-----------|---|
| n | The requested privilege level. |
| str | An optional string variable containing the password for that level. |

In the following example, 3 is the requested privilege level and ABCD is the password for that level.

```
150 CHG_PRIV 3, "ABCD"
```

No password is supplied in the next example.

```
200 CHG_PRIV 2
```

In either case, an ACK/NACK message is returned from the Service Request Processor (SRP) to the MegaBasic task, indicating a grant or refusal to grant the requested privilege level. The returned message may be obtained by the MegaBasic program through the use of the PCM extension **PROCESS_MESSAGE** procedure, either through a MegaBasic interrupt or through periodic use of the **PROCESS_MESSAGE** statement. Section 8 of this chapter contains a sample program which uses a MegaBasic interrupt to determine whether the **CHG_PRIV** command succeeded. The **GEN_TEST.PGM** package, provided with TERME, also contains an example using the **PROCESS_MESSAGE** statement.

Note

When **CHG_PRIV** fails because an invalid password was provided, a Password access failed fault is logged to the PLC fault table.

For PCM Release 3.00 and later, a simpler method is available for changing the privilege level. Simply include the following MegaBasic statements in your program:

```
150 Open #5, "CPU:#5"
160 Ioctl #5, "PASSWD"
170 Close #5
```

The correct password string for the desired privilege level should be substituted for PASSWD.

SMSG_WTEXT

SMSG_WTEXT is a general-purpose procedure that makes requests of the Service Request Processor (SRP) by sending a message with text string to the SRP.

The format for using the **SMSG_WTEXT** procedure is:

```
xxx SMSG_WTEXT req_code, acc_code, offset, length, handle
```

| Parameter | Description |
|--------------|---|
| Request Code | The request code for the particular PLC reference data type (see below). |
| Access Code | The access code for this user reference type (see below). |
| Offset | The zero-based offset from the beginning of the specified user reference data. |
| Length | The number of data words to transfer. |
| Handle | A user identification for this transfer. The procedure sends a message and a text string defined as SMSG_TXT\$. |

Note

The current size of the data in the text string **SMSG_TXT\$** must be in the range 0 through 256 bytes.

The SRP response message from a call to **SMSG_WTEXT** may be obtained by using the **PROCESS_MESSAGE** statement, which can be used periodically, or by an interrupt procedure invoked by the response.

Before using **SMSG_WTEXT**, the PLC program name, which is the same as the Logicmaster 90 folder name used to create the program, and block name must first be put into the shared string variable **SMSG_TXT\$**. If %L data is to be accessed, its sub-block name must be added to **SMSG_TXT\$** starting at character position 9.

The request and access codes for PLC memory types are:

Table 5-4. Request and Access Codes for PLC Memory Types

| User Reference | Request Code | Access Code | Use |
|---------------------|------------------------|------------------|-----------|
| %L (Series90-70) | R_PBMEM\$ ² | ACC_CODES\$(1:) | Read %L |
| | W_PBMEM\$ ² | ACC_CODES\$(1:) | Write %L |
| %P (Series90-70) | R_TMEM\$ ¹ | ACC_CODES\$(2:) | Read %P |
| | W_TMEM\$ ¹ | ACC_CODES\$(2:) | Write %P |
| %I | R_SMEM\$ ³ | ACC_CODES\$(3:) | Read %I |
| | W_SMEM\$ ³ | ACC_CODES\$(3:) | Write %I |
| %Q | R_SMEM\$ ³ | ACC_CODES\$(4:) | Read %Q |
| | W_SMEM\$ ³ | ACC_CODES\$(4:) | Write %Q |
| %R | R_SMEM\$ ³ | ACC_CODES\$(5:) | Read %R |
| | W_SMEM\$ ³ | ACC_CODES\$(5:) | Write %R |
| %T | R_SMEM\$ ³ | ACC_CODES\$(6:) | Read %T |
| | W_SMEM\$ ³ | ACC_CODES\$(6:) | Write %T |
| %M | R_SMEM\$ ³ | ACC_CODES\$(7:) | Read %M |
| | W_SMEM\$ ³ | ACC_CODES\$(7:) | Write %M |
| %AI | R_SMEM\$ ³ | ACC_CODES\$(8:) | Read %AI |
| | W_SMEM\$ ³ | ACC_CODES\$(8:) | Write %AI |
| %AQ | R_SMEM\$ ³ | ACC_CODES\$(9:) | Read %AQ |
| | W_SMEM\$ ³ | ACC_CODES\$(9:) | Write %AQ |
| %SA | R_SMEM\$ ³ | ACC_CODES\$(10:) | Read %SA |
| %SB | R_SMEM\$ ³ | ACC_CODES\$(11:) | Read %SB |
| %SC | R_SMEM\$ ³ | ACC_CODES\$(12:) | Read %SC |
| %S | R_SMEM\$ ³ | ACC_CODES\$(13:) | Read %S |

¹ R_TMEM = Read task memory;

W_TMEM = Write task memory.

² R_PBMEM = Read program block memory;

W_PBMEM = Write program block memory.

³ R_SMEM = Read system memory;

W_SMEM = Write system memory.

In the following example, the **MSG_WTEXT** procedure is used to read %L20. **GEN_MSG** is the PLC program name/folder name, and **LMEM** is the program subblock name associated with the particular %L memory that is to be read. **R_PBMEM\$** is the request code used to read %L memory. **ACC_CODE\$(1:)** is used for %L memory, 19 is the offset from the beginning of %L memory to %L20, 1 is the number of %L registers to read, and 123 is a code that the MegaBasic program can use to recognize the PLC response message for this request if **PROCESS_MESSAGE** returns more than one message. Note that **MSG_TEXT\$** is shared from the **GENERIC.PGM** package and is the actual text string sent by **MSG_WTEXT**

```
220 MSG_TEXT$(1:8) = "GEN_MSG" + chr$(0)
230 MSG_TEXT$(9:5) = "LMEM" + chr$(0)
330 MSG_WTEXT R_PBMEM$, ACC_CODE$(1:), 19, 1, 123
```

In the next example, the **MSG_WTEXT** procedure is used to write %P15-%P19. **GEN_MSG** is the PLC program name/folder name. **W_TMEM\$** is the request code needed to write %P memory. **ACC_CODE\$(2:)** is used for %P memory, 14 is the offset from the beginning of %P memory to %P15, 5 is the number of %P registers to write, and 26 is the handle. Note that a block name is not necessary for %P memory.

```
220 MSG_TEXT$(1:8) = "GEN_MSG" + chr$(0)
225 MSG_TEXT$(9:) = SEND_P$
330 MSG_WTEXT W_TMEM$, ACC_CODE$(2:), 14, 5, 26
```

Section 4: Access to PLC Fault Tables and PLC Status

READ_FLT.PGM provides access to the PLC and I/O fault tables from a MegaBasic program. In addition, **PRN_FLT.PGM** and **TEST_FLT.PGM** explain how to use the fault information supplied by **READ_FLT.PGM**. **PRN_FLT.PGM** provides several routines that break down and print information contained in the fault records. **TEST_FLT.PGM** explains how to use the functions and procedures from the other two files to read and display the fault table information.

To use the functions, procedures, and data structure definitions supplied in **READ_FLT.PGM**, add the line:

```
xxx Access "READ_FLT.pgm"
```

at the beginning of a MegaBasic program or in the PROLOGUE section of a user-written package. The **READ_FLT.PGM** file must be loaded to the PCM RAM Disk before running the MegaBasic program which accesses it. The file is located in the \PCOP or \PCOP\EXAMPLES.PCM directory of your PC hard disk, depending on which release of PCOP or TERMF you use.

Note

In PCOP version 2.04 and earlier, **READ_FLT.PGM**, **PRN_FLT.PGM**, and **TEST_FLT.PGM** are called **RD_FLT.PGM**, **PR_FLT.PGM**, and **RP_TEST.PGM**, respectively.

The functions and procedures found in **READ_FLT.PGM** are listed below:

Table 5-5. READ_FLT.PGM Functions and Procedures

| Function/Procedure | Description |
|--------------------|---|
| READ_FAULT_TBL | A procedure that reads the table header information and zero (0) or more fault records from the I/O or PLC fault table. |
| RDEL_FAULT_TBL | A procedure that reads and deletes the first fault record in either fault table. |
| FLT_PRESENT%() | A function that reads PLC short status information and returns the fault table present boolean from the PLC status word for either fault table. |
| FLT_CHANGED%() | A function that reads PLC short status information and returns the fault table changed boolean from the PLC status word for either fault table. |
| WORD%() | A function that converts the first two bytes of a string to an integer. |

Although some Series 90 PLC services require the requester to be a programmer (i.e., a device authorized to change the PLC program), none of the procedures or functions in this section requires it. As a result, there is no interference between these functions or procedures and Logicmaster 90 software, except that the use of **RDEL_FAULT_TBL** will cause faults to disappear from the Logicmaster 90 fault table displays, and the Logicmaster 90 clear fault table function may cause the MegaBasic program to miss faults.

Several shared structure definitions and constants are also defined in `READ_FLT.PGM`. These include:

Table 5-6. READ_FLT.PGM Shared Definitions and Constants

| Constant/Structure | Description |
|-----------------------|--|
| IO_FLT_TBL | A constant (1) used to indicate the selection of the I/O fault table in the above functions and procedures. These functions and procedures default to the I/O fault table when the table selection parameter is omitted. |
| PLC_FLT_TBL | A constant (2) used to indicate the selection of the PLC fault table in the above functions and procedures. |
| PLC_FAULT_HDR\$ | A string variable of length 12 containing the header information from the PLC fault table. This string is filled in as a result of a call to the <code>READ_FAULT_TBL</code> procedure that explicitly selects the <code>PLC_FLT_TBL</code> . |
| IO_FAULT_HDR\$ | A string variable of length 12 containing the header information from the I/O fault table. This string is filled in as a result of a call to the <code>READ_FAULT_TBL</code> procedure that selects the default <code>IO_FLT_TBL</code> . |
| SHORT_STATUS\$ | A string variable of length 12 containing the data returned by a short status request to the PLC CPU. This variable is updated by every call to <code>FLT_PRESENT%</code> or <code>FLT_CHANGED%</code> and may also be updated by a <code>SYSREAD</code> . |
| CLEAR_TSS | A structure definition of the header records from either fault table. |
| SS_NUM_CONTROL_PROGSS | A structure definition of the short status record. |
| TS_SEC\$ | A structure definition of time stamp fields. |
| IO_FTS | A structure definition of the I/O fault table records. |
| IOFA_SEGS | A structure definition of the I/O reference address field of the I/O fault record. |
| IOFA_RACKS | A structure definition of the I/O fault address field of the I/O fault record. |
| PLC_FTS | A structure definition of the PLC fault table records. |
| PLCFA_RACKS | A structure definition of the PLC fault address field of the PLC fault record. |

These structures are predefined in `READ_FLT.PGM`. Any field of these structures can be accessed by name; for example:

```
IO_FLT_RECS$(I%*42:42).IO_FLT_ADD$.IOFA_RACK$
```

Other examples can be found in `PRN_FLT.PGM` and `TEST_FLT.PGM`. `PRN_FLT.PGM` shows the use of the structures defined in `READ_FLT.PGM`. Refer to a listing of `PRN_FLT.PGM` for examples of the use of these structures. `TEST_FLT.PGM` provides an example of the use of `READ_FLT.PGM` and `PRN_FLT.PGM`.

READ_FAULT_TBL

The Read Fault Table (**READ_FAULT_TBL**) procedure enables entries from either fault table to be read into a user-supplied string. The read performed by **READ_FAULT_TBL** is not destructive, so that other users (e.g., Logicmaster 90 software) may read the same fault table entries.

The user-supplied string must be at least 42 bytes long (one fault record length). The maximum number of fault records that may be read from the I/O fault table is 32. Up to 16 fault records may be read from the PLC fault table.

The actual number of faults read from the table can be determined by looking at the **NUM_READ\$** field of the appropriate header string (**IO_FAULT_HDR\$.NUM_READ\$** or **PLC_FAULT_HDR\$.NUM_READ\$**, depending on the selected fault table) after using the **READ_FAULT_TBL** procedure.

The format for using the **READ_FAULT_TBL** procedure is:

```
xxx READ_FAULT_TBL str, select, start, num
```

| Parameter | Description |
|-----------|---|
| String | The string where the procedure puts the fault record(s) read from the PLC CPU. |
| Select | IO_FLT_TBL or PLC_FLT_TBL . The default is IO_FLT_TBL . |
| Start | The starting fault record number to begin reading the fault table. The default is 1, the first entry. |
| Number | The number of fault records to read. The default is 0, which returns only the fault table header. |

Note

All parameters except **String** are optional.

In the following example, fault records are read from the I/O fault table, starting with the first record and reading up to 32 records. The fault records are placed in the string **IO_FLTS\$**.

```
50 Access "READ_FLT.pgm"
100 Dim IO_FLTS$(32*42)
2000 READ_FAULT_TBL IO_FLTS$, IO_FLT_TBL, 1, 32
```

In the next example, only the I/O fault table header string, **IO_FAULT_HDR\$**, is read.

```
50 Access "READ_FLT.pgm"
100 Dim TEMP$(42)
300 READ_FAULT_TBL TEMP$
```

RDEL_FAULT_TBL

The Read and Delete Fault Table (**RDEL_FAULT_TBL**) procedure reads and deletes the first entry in either fault table. The deleted first entry is returned in a user-defined string. The string must be at least 42 bytes long (one fault record length).

The format for using the **RDEL_FAULT_TBL** procedure is:

```
xxx RDEL_FAULT_TBL str, select
```

| Parameter | Description |
|-----------|--|
| String | The string where the procedure puts the fault record, if any, read from the PLC CPU. |
| Select | IO_FLT_TBL or PLC_FLT_TBL . The select parameter is optional. The default is IO_FLT_TBL . |

In this example, the first fault record in the PLC fault table is placed in the string **FLT\$** and then deleted.

```
50 Access "READ_FLT.pgm"
100 Dim FLT$(42)
400 RDEL_FAULT_TBL FLT$, PLC_FLT_TBL
```

In the next example, the first fault record in the I/O fault table is placed in the string **FLT\$** and then deleted. Note the default selection of the I/O fault table.

```
50 Access "READ_FLT.pgm"
100 Dim FLT$(42)
600 RDEL_FAULT_TBL FLT$
```

FLT_PRESENT%

The Fault Present (**FLT_PRESENT%**) function checks either fault table to see if there are faults present in the table. The return value of the function is a boolean (true/false/1/0) value. The return value is true if faults are present in the selected table and false if there are no faults.

The format for using the **FLT_PRESENT%** function is:

```
xxx result = FLT_PRESENT%(select)
```

| Parameter | Description |
|-----------|------------------------------------|
| Select | IO_FLT_TBL or PLC_FLT_TBL |
| Result | A boolean true/false or 1/0 value. |

The following example uses the **FLT_PRESENT%** function:

```
50 Access "READ_FLT.pgm"
100 If FLT_PRESENT%(PLC_FLT_TBL) then [
110   Print "There are faults in the PLC fault table."
120 ] else [
130   Print "The PLC fault table is empty."
140 ]
```

This time the **FLT_PRESENT%** function is used to check the I/O fault table:

```
50 Access "READ_FLT.pgm"
100 If FLT_PRESENT% then [
110   Print "There are faults in the I/O fault table."
120 ] else [
130   Print "The I/O fault table is empty."
140 ]
```

Note

When the optional parameter for selection of the fault table is omitted, the parentheses around the parameter list must also be omitted.

FLT_CHANGED%

The Fault Changed (**FLT_CHANGED%**) function checks either fault table to see if faults have been added to or removed from the table since the last time it was checked. The return value of the function is a boolean (true/false, 1/0) value. The return value is true if the selected fault table contents have changed since the last check of the fault table and false if there have been no changes.

The format for using the **FLT_CHANGED%** function is:

```
xxx result = FLT_CHANGED%(select)
```

| Parameter | Description |
|-----------|--------------------------------------|
| Select | FLT_TBL or PLC_FLT_TBL |
| Result | A boolean (true/false or 1/0) value. |

The following example uses the **FLT_CHANGED%** function:

```
50 Access "READ_FLT.pgm"
100 Dim PLC_FLTS$(16*42)
3000 If FLT_CHANGED%(PLC_FLT_TBL) then [
3100     READ_FAULT_TBL PLC_FLTS$, PLC_FLT_TBL, 1, 16
3200 ]
```

An example using the **FLT_CHANGED%** function to check the I/O fault table is:

```
50 Access "READ_FLT.pgm"
100 Dim IO_FLTS$(32*42)
3000 If FLT_CHANGED% then [
3100     READ_FAULT_TBL IO_FLTS$, IO_FLT_TBL, 1, 32
3200 ]
```

Note

When the optional parameter for selection of the fault table is omitted, the parentheses around the parameter list must also be omitted.

WORD%

The **WORD%** function provides access to 2-byte integer fields present in the fault records, header records and short status record. The return value of the function is the integer value of the first two bytes of the string parameter passed to the function.

The format for using the **WORD%** function is:

```
xxx result = WORD%(str)
```

| Parameter | Description |
|-----------|--|
| String | A string with current size of at least 2. |
| Result | An integer value containing the first two bytes of the string combined into a single word. |

In the following example, the **WORD%** function converts the PLC status word field of the short status record to an integer. The integer is then formatted as a 16-bit hex number and printed. Note that %"4h4" is a print format that displays 16-bit values as 4 digit hex numbers, including leading zeroes if necessary. (More information on the PLC status word is provided later in this section.)

```
50 Access "READ_FLT.pgm"
600 SYSREAD SHORT_STATUS$
610 Print %"4h4",WORD%(SHORT_STATUS$.SS_PLC_STATUS$)
```

In the next example, the header record for the I/O fault table is read. The number of faults field is converted by **WORD%** function and then printed.

```
50 Access "READ_FLT.pgm"
100 Dim TEMP$(42)
300 READ_FAULT_TBL TEMP$
310 Print WORD%(IO_FAULT_HDR$.NUM_FLTSS$)," I/O faults present."
```

Fault Table Header Records

When using the **READ_FAULT_TBL** procedure, a header record is returned through the **IO_FAULT_HDR\$** or **PLC_FAULT_HDR\$** string variables. Header records are the same for both fault tables and contain the following information:

| Field | Description |
|--------------------|--|
| CLEAR_TSS | A six-byte field containing the time stamp when the fault table was last cleared. (Refer to the discussion below of time stamp subrecords for a definition of this field.) |
| FLTS_SINCE_CLEAR\$ | A two-byte integer (WORD%) field containing a count of the number of faults that have occurred since the last time the fault table was cleared. |
| NUM_FLT\$ | A two-byte integer (WORD%) field containing a count of fault records presently in the fault table. |
| NUM_READ\$ | A two-byte integer (WORD%) field containing a count of the actual number of fault records returned as a result of the READ_FAULT_TBL call. |

PLC Fault Table Records

When requesting a non-zero number of fault records from the PLC fault table, you may receive one or more PLC fault records. The actual number of records returned may be determined from the **NUM_READ\$** field of the **PLC_FAULT_HDR\$** record described above. Each PLC fault record consists of 42 bytes with the following format:

| Field | Description |
|-----------------------|---|
| PLC_FTS | A one-byte flag indicating the type of fault record and how much of the PLC_FLT_SPEC\$ field is valid data. The codes used for this field are: 0 = Fault record is a PLC fault record; only the first 8 bytes of PLC_FLT_SPEC\$ data are valid. 1 = Fault record is a PLC fault record; all 24 bytes of PLC_FLT_SPEC\$ data are valid. |
| Spare/reserved | Three bytes. |
| PLC_FLT_ADD\$ | A four-byte field containing the address of the module that reported the fault. (See discussion below of PLC fault addresses.) |
| PLC_FLT_GRP\$ | A one-byte integer specifying the fault group * of the fault. |
| PLC_FLT_ACT\$ | A one-byte integer specifying the action code * of the fault. |
| PLC_FLT_ERROR_CODES\$ | A two-byte integer (WORD%) specifying the actual error code * for the fault. |
| PLC_FLT_SPEC\$ | 8 or 24 bytes of fault-specific or extra fault data (rarely used). |
| PLC_FLT_TSS | A six-byte field containing the time stamp when the fault occurred. See discussion below of time stamp subrecords. |

* Refer to the *Series 90-70 Programmable Controller Installation Manual*, GFK-0262, for additional terms associated with Series 90 faults.

I/O Fault Table Records

When a non-zero number of fault records is requested from the I/O fault table, you may receive one or more I/O fault records. The actual number of records returned may be determined from the **NUM_READ\$** field of the **IO_FAULT_HDR\$** record described above. Each I/O fault record contains 42 bytes and has the following format:

| Field | Description |
|---------------|--|
| IO_FTS | A one-byte flag indicating the type of fault record and how much of the IO_FLT_SPEC\$ field is valid data. The codes used for this field are: 2 = Fault record is an I/O fault record. Only the first 5 bytes of IO_FLT_SPEC\$ data are valid. 3 = Fault record is an I/O fault record. All 21 bytes of IO_FLT_SPEC\$ data are valid. |
| IO_REF_ADD\$ | A three-byte field describing the I/O reference address where the fault was reported. See the discussion of the I/O reference address sub-record below. |
| IO_FLT_ADD\$ | A six-byte field describing the physical address of the fault. See the discussion of the I/O fault address sub-record below. |
| IO_FLT_GRP\$ | A one-byte integer specifying the fault group * of the fault. |
| IO_FLT_ACT\$ | A one-byte integer specifying the action code * of the fault. |
| IO_FLT_CAT\$ | A one-byte integer specifying the category * of the fault. |
| IO_FLT_TYPE\$ | A one-byte integer specifying the type code * of the fault. |
| IO_FLT_DESC\$ | A one-byte integer specifying the description code * of the fault. |
| IO_FLT_SPEC\$ | 5 or 21 bytes of fault-specific or extra fault data (rarely used). |
| IO_FLT_TSS | A six-byte field containing the time stamp when the fault occurred. See the discussion below of the time stamp subrecord. |

* Refer to the *Series 90-70 Programmable Controller Installation Manual*, GFK-0262, for additional terms associated with Series 90 faults.

Short Status Records

When using the **FLT_PRESENT%** or **FLT_CHANGED%** function, or when using **SYSREAD** to update **SHORT_STATUS\$** or a variable **SYSLINK**ed to **#SSTAT**, a short status record is created:

| Field | Description |
|-------------------------|--|
| SS_NUM_CONTROL_PROG\$\$ | A one-byte integer containing the number of control program tasks currently defined in the PLC CPU (range = 0 to 8). |
| SS_PROGRAMMER_FLAG\$\$ | A set of boolean flags indicating which control program tasks have programmers currently attached to them. |
| Spare/Reserved | 4 bytes. |
| SS_CONTROL_PROG_NUM\$ | A one-byte integer number indicating which control program the MegaBasic program is currently attached to (range = -1 to 7, -1 means not currently attached to a control program). |
| SS_PRIV_LEVEL\$ | The current privilege level of the MegaBasic program for accessing memory in the PLC CPU. |
| SS_SWEEP_TIMES | A two-byte integer (WORD%) containing the time interval from the end of the second most recent PLC sweep to the end of the most recent sweep. This value is in 100 microsecond increments; it is zero when the PLC is not running a user program. |
| SS_PLC_STATUS\$ | A two-byte set of boolean flags and bit fields describing the current status of the PLC CPU. Using the MegaBasic bit numbering convention, the following bit fields are defined: |
| | 0:1 = Enable/Disable switch setting. 1 = Disable outputs. 0 = Enable outputs. |
| | 1:1 = Programmer attachment present flag. 1 = There is a programmer attachment for the indicated control program number. 0 = There is no programmer attached to that control program. |
| | 2:1 = I/O fault table entry present flag. This flag is also returned by the FLT_PRESENT% function. 1 = There is a fault record in the I/O fault table. 0 = There is no fault record in the I/O fault table. |
| | 3:1 = PLC fault table entry present flag. This flag is also returned by the FLT_PRESENT% (PLC_FLT_TBL) function. 1 = There is a fault record in the PLC fault table. 0 = There is no fault record in the PLC fault table. |
| | 4:1 = I/O fault entry changed flag. This flag is also returned by the FLT_CHANGED% function. 1 = I/O fault table has changed since the last time the table was read. 0 = Table has not changed. |

| | |
|--|---|
| | <p>5:1 = PLC fault entry changed flag. This flag is also returned by the FLT_CHANGED% (PLC_FLT_TBL) function.</p> <p>1 = PLC fault table has changed since the last time the table was read.</p> <p>0 = Table has not changed.</p> |
| | <p>6:1 = Constant sweep mode setting.</p> <p>1 = Constant sweep mode of PLC CPU operation has been enabled.</p> <p>0 = Constant sweep mode is disabled.</p> |
| | <p>7:1 = Oversweep flag (valid only if constant sweep mode is enabled).</p> <p>1 = Constant sweep value was exceeded on the last sweep.</p> <p>0 = Constant sweep value was not exceeded.</p> |
| | <p>8:4 = Current PLC CPU state (as opposed to switch settings or commanded state). Possible values are:</p> <p>0 = PLC running, I/O enabled.</p> <p>1 = PLC running, I/O disabled.</p> <p>2 = PLC stopped.</p> <p>3 = PLC stopped due to a fatal fault.</p> <p>4 = PLC halted.</p> <p>5 = PLC suspended.</p> <p>6 = PLC stopped, I/O enabled.</p> |
| | <p>12:2 = Spare reserved.</p> |
| | <p>14:1 = Program/Configuration/Symbol status table changed flag.</p> <p>1 = A change has been made to the PLC user program, configuration data or symbol status table. The latter requires a new resolve of any symbols currently resolved by the MegaBasic program. This function is not presently used by the system.</p> <p>0 = No changes.</p> |
| | <p>15:1 = RUN/STOP switch setting from the front panel.</p> <p>1 = RUN.</p> <p>0 = STOP.</p> |

Time Stamp Subrecords

Time stamps are used in the fault header records to indicate when the fault table was last cleared. They are also used in fault records to indicate when the faults occurred. The structure of these time stamps is explained in the following table:

| Time Stamp | Description |
|------------|--|
| TS_SECS | A one-byte BCD integer (range 0-59) indicating the seconds count. |
| TS_MIN\$ | A one-byte BCD integer (range 0-59) indicating the minutes count. |
| TS_HOURS | A one-byte BCD integer (range 0-23) indicating the hour count. |
| TS_DAYS | A one-byte BCD integer (range 1-31) indicating the day of the month. |
| TS_MON\$ | A one-byte BCD integer (range 1-12) indicating the month. |
| TS_YEARS | A one-byte BCD integer (range 0-99) indicating the year. |

PLC Fault Address Subrecords

In the PLC fault records, the fault address has this structure:

| Integer | Description |
|--------------|---|
| PLCFA_RACK\$ | A one-byte integer indicating the rack number of the source of the fault. |
| PLCFA_SLOTS | A one-byte integer indicating the slot number of the source of the fault. |
| PLCFA_UNITS | A one-byte integer indicating the unit, task, or connection point of the source of the fault. |

I/O Reference Address Subrecords

In the I/O fault records, the reference address has this structure:

| Integer | Description |
|------------|--|
| IORA_SEGS | A one-byte user reference indicator with the following typical decimal values: 16 or 70 = Input table (%I). 18 or 72 = Output table (%Q). 10 = Analog input table (%AI). 12 = Analog output table (%AQ). |
| IORA_ADD\$ | A two-byte integer (WORD%) with range 1 to maximum table size. |

I/O Fault Address Subrecords

In the I/O fault records, the fault address has this structure:

| Integer | Description |
|------------------|---|
| IOFA_RACK\$ | A one-byte integer indicating the rack number for the source of the fault. |
| IOFA_SLOTS | A one-byte integer indicating the slot number for the source of the fault. |
| IOFA_BUSS | A one-byte integer indicating the bus number for the source of the fault. |
| IOFA_BUS_ADD\$ | A one-byte integer indicating the bus address for the source of the fault. |
| IOFA_PT_OFFSET\$ | A two-byte integer (WORD%) indicating the offset for the source of the fault. |

If the fault is slot-related, the lower level parts of the fault address (bus, bus address, and point offset) may be set to -1.

Known Problems with Fault Table Access

There are several known problems with fault table access in early versions of the PLC CPU firmware for the Series 90-30 and 90-70 PLCs.

| PLC | Description |
|------------------|---|
| Series 90-70 PLC | The fault entry present bits in the PLC status word are not cleared if the table is emptied by doing repeated read and delete operations (RDEL_FAULT_TBL). |
| Series 90-30 PLC | The fault entry changed bits are not cleared after reading the fault table(s), except for the programmer attachment (READ_FAULT_TBL). |
| Series 90-30 PLC | The PLC CPU returns the second fault in the table when doing a read and delete sequence, instead of the first fault in the table (RDEL_FAULT_TBL). |
| Series 90-30 PLC | The PLC CPU does not return any response at all to a read and delete sequence if the requested fault table is empty. |

Section 5: Gathering PLC Information from MegaBasic Programs

The **UTILITY.PGM** package supplies several procedures, as well as several data buffers and flag variables, for gathering system information from the Series 90 CPU.

To use the procedures, data buffers and flag variables supplied in this **UTILITY.PGM** package, add the line:

```
xxx Access "utility.pgm"
```

to the beginning of the MegaBasic program or in the PROLOGUE section of a user written package. The file **UTILITY.PGM** should have been loaded to the PCM RAM Disk before running the MegaBasic program which accesses it. The file will be located in the \PCOP or \PCOP\EXAMPLES.PCM directory of your hard disk, depending on which version of PCOP or TERMF you use.

During the PROLOGUE processing that takes place as part of the first access to the **UTILITY.PGM** package, the following interrupts are linked to procedures in the package: **BKP_MSG**, **TIMER1**, **TIMER2**, **TIMER3**, **TIMER4**, and **TIMER5**. The timers associated with these five timer interrupts are also stopped.

If the user changes these interrupt linkages, the original settings can be restored through the **UTILITY_INIT** procedure described below.

Some of the procedures in the **UTILITY.PGM** package support a NOWAIT operation. As a result of the NOWAIT mode of operation, the requested data may not be available immediately after returning from the called procedure. However, there is a global status variable (typically "..._VALID") that indicates when the data has arrived via the **BKP_MSG** interrupt.

The NOWAIT operation also allows some of the procedures in the package to obtain the requested data repetitively. When the procedure has setup a repetitive read operation, one of the timer interrupts listed above may be used in addition to the **BKP_MSG** interrupt. The data valid flag is incremented every time new data arrives and rolls over from 255 to 1. The only time that the data valid flag is a zero (a logical false) is when the first transfer of the data after the procedure call has not yet taken place.

The procedures found in the **UTILITY.PGM** package are summarized in the following table.

Table 5-7. UTILITY.PGM Package Procedures

| Procedure | Description |
|------------------------|---|
| READ_PLC_STATUS | A procedure used to get the PLC status word. The optional parameters for this procedure indicate the method of return: wait for completion of the read operation (default) or return immediately (NOWAIT) with the data valid flag being used to indicate when the actual data has arrived. In addition, for NOWAIT, the command may be automatically repeated at a specified frequency, as fast as possible, or no repeats at all. The PLC status word is placed in the integer variable <code>PLC_STATUS_WORD</code> and the <code>PLC_STATUS_WORD_VALID</code> integer variable is used to indicate when the status word is updated. |
| READ_PLC_TIME_AND_DATE | <p>A procedure used to get the time and date information from the PLC CPU. The time value is formatted (hh:mm:ss format) into the 8-byte string variable <code>PLC_TIME</code>, and the date is formatted (mm/dd/yy format) into the 8-byte string variable <code>PLC_DATE</code>. The integer variable <code>PLC_TIME_AND_DATE_VALID</code> is used to indicate when the time and date variables have been updated.</p> <p>Optional parameters for this procedure are the same as those previously described for <code>READ_PLC_STATUS</code>.</p> |
| READ_PLC_RUN_STATUS | <p>A procedure that gets the PLC run status from the PLC status word and sets the <code>PLC_RUN_STATUS</code> integer variable according to the value found in the PLC status word as follows:</p> <ul style="list-style-type: none"> 0 = PLC run enabled. 1 = PLC run disabled. 2 = PLC stopped. 3 = PLC stopped/faulted. 4 = PLC halted. 5 = PLC suspended. 6 = PLC stopped I/O enabled. <p>The integer variable <code>PLC_RUN_STATUS_VALID</code> is used to indicate when the PLC run status variable has been updated.</p> <p>Optional parameters for this procedure are the same as those previously described for <code>READ_PLC_STATUS</code>.</p> |
| READ_PLC_CPU_ID | <p>A procedure that gets the PLC ID string from the PLC CPU long status information (<code>#LSTAT</code>). The length of the string is adjusted to permit proper printing of the PLC CPU ID (ie, it is converted from zero-terminated ASCII format to MegaBasic string format). The string variable <code>PLC_CPU_ID</code> contains the ID string after it is read, and the integer variable <code>PLC_CPU_ID_VALID</code> indicates when the ID has been read.</p> <p>The only optional parameter for this procedure is to select NOWAIT mode, if desired.</p> |

Table 5-7. UTILITY.PGM Package Procedures (cont'd)

| Procedure | Description |
|--------------------|--|
| CHECK_CPU_HEALTH | <p>A procedure that performs a short status read and, if the PLC CPU responds properly, the integer variable <code>CPU_RESPONDING</code> is set to true. If the PLC CPU does not respond properly after three (3) tries, the <code>CPU_RESPONDING</code> integer variable is set to false. Every time the CPU responds, the local time and date is recorded in string variables <code>TIME_OF_LAST_CPU_RESP</code> and <code>DATE_OF_LAST_CPU_RESP</code>. The validity of the time and date strings is indicated by the integer variable <code>TIME_AND_DATE_OF_LAST_CPU_RESP_VALID</code>.</p> <p>The only optional parameter for this procedure specifies how often to check the PLC CPU's health, specified in milliseconds. The default selection is to stop checking the CPU's health. Unlike other procedures, calling this procedure with no parameters does not perform a one-time check of the CPU's health.</p> |
| READ_PLC_FAULT_BIT | <p>A procedure used to obtain the status of a fault summary bit from the PLC CPU. Faultsummary bits indicate the presence of one or more faults in a specified rack, slot, bus, or module.</p> <p>Optional parameters for this procedure include four parameters to indicate the desired fault summary bit and two parameters for the selection of NOWAIT and the repetition rate. The default selection is to return the rack 0 fault summary bit waiting for the response and to do the transfer only once.</p> |
| UTILITY_INIT | <p>A procedure used to (re)initialize the interrupt linkages used for the various timers and backplane interrupts for the proper functioning of the rest of the procedures in this package. This procedure is called automatically in the PROLOGUE section of the package but may also be called at any time to re-establish the interrupt routine linkages and timer states after the execution of other packages/routines using the same interrupt(s) or timer(s) as the <code>UTILITY.PGM</code> package.</p> |

Shared constants and variables, defined in the UTILITY.PGM package, are described in the following table:

Table 5-8. UTILITY.PGM Package Shared Constants and Variables

| Name | Description |
|-----------------|---|
| TRUE | A constant (= 1) used to indicate a logical true condition. |
| FALSE | A constant (= 0) used to indicate a logical false condition. |
| PLC_STATUS_WORD | <p>An integer variable that contains the PLC status word, as last read from the PLC CPU. This word is a set of binary flags and bit fields that indicate various information about the PLC CPU's state. When numbered so that the LSB is bit 0, the following information is available:</p> <ul style="list-style-type: none"> 0 = Oversweep flag, valid only if constant sweep mode is enabled. 1 = Constant sweep value was exceeded on last sweep. 0 = Constant sweep value was not exceeded. |

Table 5-8. UTILITY.PGM Package Shared Constants and Variables (cont'd)

| | |
|-----------------------------|---|
| PLC_STATUS_WORD (cont'd) | 1 = Constantsweep mode setting. 1 = Constant sweep mode is enabled. 0 = Constant sweep mode is disabled. |
| | 2 = PLC fault entry changed flag. 1 = PLC fault table has changed since last time the table was read. 0 = Table has not changed. This value is returned by the FLT_CHANGED% (PLC_FLT_TBL) function. |
| | 3 = I/O faultentry changed flag. 1 = I/O fault table has changed since last time the table was read. 0 = Table has not changed. This value is returned by the FLT_CHANGED% function. |
| | 4 = PLC fault table entry present flag. 1 = There is a fault record in the PLC fault table. 0 = There is no fault record in the PLC fault table. This value is returned by the FLT_PRESENT% (PLC_FLT_TBL) function. |
| | 5 = I/O faulttableentry present flag 1 = There is a fault record in the I/O fault table. 0 = There is no fault record in the I/O fault table. This value is returned by the FLT_PRESENT% function. |
| | 6 = Programmer attachment present flag. 1 = There is a programmer attachment for the indicated control program number. 0 = No programmer is attached to that control program. |
| | 7 = Enable/Disableswitchsetting. 1 = Outputs are disabled. 0 = Outputs are enabled. |
| | 8 = RUN/STOP switch setting from the front panel. 1 = RUN. 0 = STOP. |
| | 9 = Symbol status table changed flag. 1 = Symbol status table has changed, requiring a new resolve of any symbols currently resolved by the MegaBasic program. <u>Note:</u> This function is not presently used by the system. |
| | 10 and 11 = Spare reserved. |
| | 12 thru 15 = Current PLC CPU state, as opposed to switch settings or commanded state. Possible values are: 0 = PLC run enabled. 1 = PLC rundisabled. 2 = PLC stopped. 3 = PLC stopped/faulted. 4 = PLC halted. 5 = PLC suspended. 6 = PLC stopped I/O enabled. |

Table 5-8. UTILITY.PGM Package Shared Constants and Variables (cont'd)

| Name | Description |
|--------------------------------------|---|
| PLC_STATUS_WORD_VALID | An integer variable that is incremented by 1 every time the PLC status word is read. The value of the variable increments up to 255 and then rolls over to 1. A zero value (false) indicates that the PLC status word has never been read. |
| PLC_TIME | A string variable of length 8 that supplies the PLC time value read as a result of the <code>READ_PLC_TIME_AND_DATE</code> procedure. This string is formatted as hh:mm:ss, with the leading digit of the hours field being set to blank if it is a zero. |
| PLC_DATE | A string variable of length 8 that supplies the PLC date value read as a result of the <code>READ_PLC_TIME_AND_DATE</code> procedure. This string is formatted as mm/dd/yy, with the leading digit of the month field being set to blank if it is a zero. |
| PLC_TIME_AND_DATE_VALID | An integer variable that is incremented by 1 every time the PLC time and date is read. The value of the variable increments up to 255 and then rolls over to 1. A zero value (false) indicates that the PLC time and date have never been read. |
| CPU_RESPONDING | An integer variable that contains the true/false flag, indicating whether the PLC CPU is responding to the check CPU health requests (ie, short status reads). |
| TIME_OF_LAST_CPU_RESP | A string variable of length 8 that records the local time (<code>time\$</code> function) at which the CPU last responded to a health check. This string is formatted as hh:mm:ss, with the leading digit of the hours field being set to blank if it is a zero. |
| DATE_OF_LAST_CPU_RESP | A string variable of length 8 that records the local date (<code>date\$</code> function) at which the CPU last responded to a health check. This string is formatted as mm/dd/yy, with the leading digit of the month field being set to blank if it is a zero. |
| TIME_AND_DATE_OF_LAST_CPU_RESP_VALID | An integer variable that indicates (true/false) whether the time and date of last CPU response are valid. Note that the check CPU health procedure must run with a rep. rate greater than zero and successfully get a response before this integer variable is set to true. |
| PLC_RUN_STATUS | An integer variable that contains the PLC run status field from the PLC status word (bits 12-15) that is the result of the <code>READ_PLC_RUN_STATUS</code> procedure. See the <code>PLC_STATUS_WORD</code> discussion for the coding of the values in this variable. |
| PLC_RUN_STATUS_VALID | An integer variable that indicates when the run status has been read/updated. The value of this variable increments up to 255 and then rolls over to 1. A zero value (false) indicates that the PLC run status has never been read. |
| PLC_CPU_ID | A string variable of maximum length 8, as a result of the <code>READ_PLC_CPU_ID</code> procedure. The actual length depends on the current length of the CPU's ID string. |
| PLC_CPU_ID_VALID | An integer variable that indicates (true/false) whether the PLC CPU ID has been read. |
| PLC_FAULT_BIT | An integer variable that indicates the presence (true) or absence (false) of a fault in the rack, slot, bus, or module selected through the <code>READ_PLC_FAULT_BIT</code> procedure. |
| PLC_FAULT_BIT_VALID | An integer variable that indicates when the fault summary bit has been read/updated. The value of this variable increments up to 255 and then rolls over to 1. A zero value (false) indicates that the fault summary bit has never been read. |

READ_PLC_STATUS

The **READ_PLC_STATUS** procedure gets the PLC status word from the short status data from the PLC CPU's Service Request Processor (SRP). This data is obtained either in a wait for data mode (default) or a NOWAIT mode. In NOWAIT mode, the user must check the **PLC_STATUS_WORD_VALID** flag to determine when the data has actually arrived in the **PLC_STATUS_WORD** variable.

Also in NOWAIT mode, the user can specify a repeat delay time so that the PLC status word is read and updated on a continuing basis. When the delay time is specified as something other than 0 (0= as fast as possible), the **TIMER1** interrupt is used to supply the delay between repeats of the read request.

The format of the **READ_PLC_STATUS** procedure call is:

```
xxx READ_PLC_STATUS wait_flag, delay
```

where both parameters are optional; and, if omitted, a single read of the PLC status word takes place in wait mode. The **wait_flag** is normally programmed with the NOWAIT constant (= 1 or true) from the **PCMEXT.PGM** package that is accessed automatically by the MegaBasic program. The delay parameter is only used if NOWAIT mode is active and is the minimum number of milliseconds that the **UTILITY.PGM** package will wait before rereading the PLC status word.

An example of the use of the **READ_PLC_STATUS** procedure is:

```
50 Access "utility.pgm"
100 READ_PLC_STATUS NOWAIT, 500
200 OLD% = PLC_STATUS_WORD_VALID
1000 Repeat
1010   While (OLD% = PLC_STATUS_WORD_VALID); Next
1020   STS% = PLC_STATUS_WORD
1030   OLD% = PLC_STATUS_WORD_VALID
1040   Print "PLC status word = ", %"4h4", STS%
1050 Next
```

where the PLC status word is being read every 1/2 second as a result of the **READ_PLC_STATUS** procedure call.

The sequence of operations is:

1. First wait for a change in the **PLC_STATUS_WORD_VALID** flag (line 1010).
2. Record the state of the **PLC_STATUS_WORD** variable (line 1020).
3. Record the state of the **PLC_STATUS_WORD_VALID** variable for waiting for a change the next time through the loop (line 1030).
4. Finally, use the recorded state of the **PLC_STATUS_WORD** in further calculations.

The reason for this particular sequence of events is to insure that all processing is done with the same sample of the PLC status word and that a new sample of the PLC status word arrives before further processing.

Another example of the **READ_PLC_STATUS** procedure is:

```
50 Access "utility.pgm"
100 READ_PLC_STATUS
110 Print "Constant sweep mode is ",
120 If (PLC_STATUS_WORD & 2) = 0 then Print "not ",
130 Print "active."
```

where a read of the PLC status word is done in wait mode so that the data is available immediately after the call to the **READ_PLC_STATUS** procedure. Then, the constant sweep mode of the PLC status word is checked, and an appropriate message, reflecting the current state of the constant sweep mode, is printed. Note that the "div 2" part of the If statement eliminates bits below the constant sweep mode bit and the "mod 2" part eliminates bits above, leaving a logical value that is only the contents of the constant sweep mode bit.

READ_PLC_TIME_AND_DATE

The **READ_PLC_TIME_AND_DATE** procedure obtains the PLC time and date in a form that is ready to print. The time is supplied in the **PLC_TIME** string variable, and the date is supplied in the **PLC_DATE** string variable.

As with the previous procedure, there are two optional parameters for the **READ_PLC_TIME_AND_DATE** procedure. The first parameter specifies the mode that the procedure is to operate in; wait mode is the default and **NOWAIT** mode is optional. When **NOWAIT** mode is active, the **PLC_TIME_AND_DATE_VALID** integer flag variable is used to determine when the time and date has been read/updated. Also in **NOWAIT** mode, a second parameter can specify the minimum delay between repeats of the **READ_PLC_TIME_AND_DATE** information. Omitting the second parameter results in a single read of the time and date, as well as turning off any previous repetitive read operation for the time and date. When the delay time is specified as something other than 0 (0= as fast as possible), the **TIMER2** interrupt is used to supply the delay between repeats of the read request.

The format of the **READ_PLC_TIME_AND_DATE** procedure call is:

```
xxx READ_PLC_TIME_AND_DATE wait_flag, delay
```

where both parameters are optional; and, if, omitted a single read of the PLC time and date takes place in wait mode. The **wait_flag** is normally programmed with the **NOWAIT** constant, if used at all. The **delay** parameter is only used if the **NOWAIT** mode is active and is the minimum number of milliseconds that the **UTILITY.PGM** package will wait before rereading the PLC time and date.

An example of the **READ_PLC_TIME_AND_DATE** procedure is:

```
50 Access "utility.pgm"
60 Access "vt100.pgm"
100 READ_PLC_TIME_AND_DATE NOWAIT, 5000
200 OLD% = PLC_TIME_AND_DATE_VALID
300 CLS
1000 Repeat
1010   While (OLD% = PLC_TIME_AND_DATE_VALID); Next
1020   Print CUR$(12, 30), PLC_DATE, "    ", PLC_TIME(1:5), CUR$(24),
1030   OLD% = PLC_TIME_AND_DATE_VALID
1040 Next
```

where the **READ_PLC_TIME_AND_DATE** procedure is used to set up a read of the time and date every 5 seconds. Then, the screen is cleared. Every time the time and date is updated, the date and the hours:minutes part of the time is displayed in the center of the VT100 style screen. Due to the slow update rate, no attempt was made to insure that the time and date string variables are time coherent. It is assumed that any processing (in this case, printing) can be done before the next update arrives.

Another example of the `READ_PLC_TIME_AND_DATE` procedure is:

```
50 Access "utility.pgm"
100 Repeat
110   READ_PLC_TIME_AND_DATE
120   If PLC_TIME <> time$(1:8) then Print "time mismatch"
130 Next
```

where the PLC time and date are read in wait mode, and then compared with the current local time (hours:minutes:seconds only). A message is printed out if they don't match. Note that the two time values generally won't match once a second at the time of the second's tick.

Caution

This operation is not a recommended procedure since reading the time of day as fast as possible from the PLC CPU may have an adverse impact on the PLC sweep time.

READ_PLC_RUN_STATUS

The **READ_PLC_RUN_STATUS** procedure allows the user to get the PLC run status from the short status data from the SRP. This data is obtained either in a wait for data mode (default) or a NOWAIT mode. In NOWAIT mode, the user must check the **PLC_RUN_STATUS_VALID** flag to determine when the data has actually arrived in the **PLC_RUN_STATUS** variable. In NOWAIT mode, the user can specify a repeat delay time so that the PLC run status is read and updated on a continuing basis. When the delay time is specified as something other than 0 (0= as fast as possible), the **TIMER3** interrupt is used to supply the delay between repeats of the read request.

The format of the **READ_PLC_RUN_STATUS** procedure call is:

```
xxx READ_PLC_RUN_STATUS wait_flag, delay
```

where both parameters are optional; and, if omitted, a single read of the PLC run status takes place in wait mode. The **wait_flag** is normally programmed with the NOWAIT constant from the **PCMEXT.PGM** package that is accessed automatically by the MegaBasic program. The delay parameter is only used if NOWAIT mode is active and is the minimum number of milliseconds that the **UTILITY.PGM** package will wait before rereading the PLC run status.

An example of the **READ_PLC_RUN_STATUS** procedure is:

```

50 Access "utility.pgm"
100 READ_PLC_RUN_STATUS NOWAIT, 500
200 OLD% = -1
1000 Repeat
1010   While (OLD% = (Let NEW% = PLC_RUN_STATUS)); Next
1020 OLD% = NEW%
1030 Print;Print date$, " ", time$, " PLC CPU ",
1040 Case begin on NEW%
1050 Case 0
1060   Print "is running with I/O enabled",
1070 Case 1
1080   Print "is running with I/O disabled",
1090 Case 2
1100   Print "is stopped with I/O disabled",
1110 Case 3
1120   Print "is stopped with I/O disabled and faults",
1130 Case 4
1140   Print "is halted with I/O disabled",
1150 Case 5
1160   Print "is suspended with I/O disabled",
1170 Case 6
1180   Print "is stopped with I/O enabled",chr$(7)*32,
1190 Case end
2000 Next
```

where the CPU run status is read every 1/2 second. Every time that it changes, a time stamped descriptive message is printed. For state 6 (stop I/O enabled) 32 <bel> characters are also sent to the terminal as a type of alarm notification.

Another example of the **READ_PLC_RUN_STATUS** procedure is:

```
50 Access "utility.pgm"
100 READ_PLC_RUN_STATUS
200 If PLC_RUN_STATUS < 2 then [
210   Print "CPU is running"
220 ] else [
230   Print "CPU is not running"
240 ]
```

where the PLC CPU run status is checked once, and an indication of the run/stopstatus of the PLC CPU program is given.

READ_PLC_CPU_ID

The **READ_PLC_CPU_ID** procedure obtains the PLC CPU ID string from the long status data from the SRP. This data is obtained either in a wait for data mode (default) or a NOWAIT mode.

In NOWAIT mode, the user must check the **PLC_CPU_ID_VALID** flag to determine when the data has actually arrived in the **PLC_CPU_ID** string variable. There is no provision for repetitively obtaining the PLC CPU ID data since it is assumed that this data would not change very often.

The format of the **READ_PLC_CPU_ID** procedure call is:

```
xxx READ_PLC_CPU_ID wait_flag
```

where the parameter is optional; and, if omitted, the read of the PLC CPU ID string takes place in wait mode. The **wait_flag** is normally programmed with the NOWAIT constant from the **PCMEXT.PGM** package that is accessed automatically by the MegaBasic program.

An example of the **READ_PLC_CPU_ID** procedure is:

```
50 Access "utility.pgm"  
100 READ_PLC_CPU_ID NOWAIT  
1000 While (not PLC_CPU_ID_VALID); Next  
1010 Print PLC_CPU_ID
```

where the PLC CPU ID string is read in NOWAIT mode. After checking that the data has arrived, the PLC CPU ID string is printed.

Another example of the **READ_PLC_CPU_ID** procedure is:

```
50 Access "utility.pgm"  
100 READ_PLC_CPU_ID  
110 Print PLC_CPU_ID
```

where the PLC CPU ID is read in wait mode and then printed.

CHECK_CPU_HEALTH

The **CHECK_CPU_HEALTH** procedure checks that the PLC CPU is still alive and talking to the PCM by reading the short status data from the SRP on a periodic basis. This data is obtained only in NOWAIT mode. The user must check the **CPU_RESPONDING** flag to determine if the CPU is still responding.

In addition, the user may check the **TIME_OF_LAST_CPU_RESP** and **DATE_OF_LAST_CPU_RESP** string variables for a time stamp of the last CPU response. These two string variables are known to be valid when the **TIME_AND_DATE_OF_LAST_CPU_RESP_VALID** logical flag variable is set to true.

The user is also permitted to specify a repeat delay time so that the PLC is checked on a continuing basis. When the delay time is specified as something greater than 0, the **TIMER4** interrupt is used to supply the delay between repeats of the read short status request. When the repeat delay is set to zero or negative, then the checking of the PLC CPU's health is stopped.

The format of the **CHECK_CPU_HEALTH** procedure call is:

```
xxx CHECK_CPU_HEALTH delay
```

where the parameter is optional; and, if omitted, the checking of the CPU's health is stopped. The delay parameter is the minimum number of milliseconds that the **UTILITY.PGM** package will wait before rechecking the PLC CPU's health.

An example of the **CHECK_CPU_HEALTH** procedure is:

```

50 Access "utility.pgm"
100 CHECK_CPU_HEALTH 1000
1000 Repeat
1010   If not CPU_RESPONDING then [
1020     Print "PLC CPU not responding to PCM"
1030     If TIME_AND_DATE_OF_LAST_CPU_RESP_VALID then [
1040       Print "CPU's last response was at",
1050       Print DATE_OF_LAST_CPU_RESP, " ", TIME_OF_LAST_CPU_RESP
1060     ] else [
1070       Print "CPU has never responded to the PCM"
1080     ]
1090   ]
2000 Next

```

where the **CHECK_CPU_HEALTH** procedure is used to setup the periodic read of the PLC CPU's short status from the SRP every second. In the main loop of the program, the **CPU_RESPONDING** flag is checked every time through the loop. If it indicates that the CPU has stopped responding, a time stamped message is put out, indicating that the CPU has stopped talking to the PCM.

Another example of the `CHECK_CPU_HEALTH` procedure is:

```
50 Access "utility.pgm"
100 PROLOGUE:
110 CHECK_CPU_HEALTH 1000
120 Return
200 EPILOGUE:
210 CHECK_CPU_HEALTH
220 Return
1000 Repeat
1010   If not CPU_RESPONDING then Print chr$(7)*32
9999 Next
```

where, in the prologue to this example package, a check of the CPU's health is set up to occur every second; and, in the epilogue, this check of the CPU's health is stopped. In the main part of the package, the `CPU_RESPONDING` flag is checked. Every time through the main loop that the CPU is not responding, 32 <bel> characters are printed.

READ_PLC_FAULT_BIT

The **READ_PLC_FAULT_BIT** procedure gets one bit of the PLC fault summary status from the SRP. This fault summary bit is obtained either in a wait for data mode (default) or a NOWAIT mode. In NOWAIT mode, the user must check the **PLC_FAULT_BIT_VALID** flag to determine when the data has actually arrived in the **PLC_FAULT_BIT** integer variable. In NOWAIT mode, the user is also permitted to specify a repeat delay time so that the selected fault summary bit is read and updated on a continuing basis. When the delay time is specified as something other than 0 (0 = as fast as possible), the **TIMER5** interrupt is used to supply the delay between repeats of the read request.

The format of the **READ_PLC_FAULT_BIT** procedure call is:

```
xxx READ_PLC_FAULT_BIT rack, slot, bus, module, wait_flag, delay
```

where all parameters are optional; and, if omitted, a single read of the rack 0 fault summary bit takes place in wait mode. The **wait_flag** is normally programmed with the NOWAIT constant from the **PCMEXT.PGM** package that is automatically accessed by the MegaBasic program. The delay parameter is only used if NOWAIT mode is active and is the minimum number of milliseconds that the **UTILITY.PGM** package will wait before rereading the PLC run status. To “skip” some of the intervening parameters before the **wait_flag** and delay values, use -1 for these parameters so that the **READ_PLC_FAULT_BIT** procedure knows what is desired.

An example of the **READ_PLC_FAULT_BIT** procedure is:

```
50 Access "utility.pgm"
100 READ_PLC_FAULT_BIT 0, -1, -1, -1, NOWAIT, 2000
200 OLD% = -1
1000 Repeat
1010   If (OLD% = (Let NEW% = PLC_FAULT_BIT)) then Next
1020   OLD% = NEW%
1030   Print date$, " ", time$, " Rack 0 fault summary bit is ",
1040   If not OLD% then Print "not ",
1050   Print "active."
2000 Next
```

where a repetitive read of the fault summary bit for rack 0 is set up to occur every two seconds. Every time it changes, a time stamp message indicating the new state is printed.

Another example of the **READ_PLC_FAULT_BIT** procedure is:

```
50 Access "utility.pgm"
100 READ_PLC_FAULT_BIT 0, 3, 1, 23
200 Print "The fault summary bit for module 23 on bus 1 of the"
210 Print " Genius Bus Controller in slot 3 is:", PLC_FAULT_BIT
```

where the fault summary bit for module 23 of bus 1 of the GBC in slot 3 of rack 0 is read and printed.

UTILITY_INIT

The **UTILITY_INIT** procedure returns all interrupt linkages and timer setups used by the **UTILITY.PGM** package to the initial state that they were upon initial access to the package. This provides a fast way to turn off all timers that may be used by the **UTILITY.PGM** package, or to make sure that the interrupt linkages are still in good condition so that the procedures in the **UTILITY.PGM** package may be used in a NOWAIT/repetitivemanner.

The format of the **UTILITY_INIT** procedure call is:

```
xxx UTILITY_INIT
```

where there are no parameters.

An example of the **UTILITY_INIT** procedure is:

```
50 Access "utility.pgm"
100 CHECK_CPU_HEALTH 1000
110 READ_PLC_FAULT_BIT 0, -1, -1, -1, NOWAIT, 2000
120 READ_PLC_RUN_STATUS NOWAIT, 500
3000 UTILITY_INIT
```

where several repetitive operations using the **UTILITY.PGM** package are started. Then, at some later time, they are all stopped by the **UTILITY_INIT** procedure call.

Another example of the **UTILITY_INIT** procedure is:

```
50 Access "utility.pgm"
100 Interrupt TIMER4, end
110 Interrupt TIMER4, DELAY_STARTUP_PROC
120 Interrupt TIMER4, on
130 TIMER 4, TSTART, 10000
3000 Def proc DELAY_STARTUP_PROC
3010 UTILITY_INIT
3020 CHECK_CPU_HEALTH 1000
3030 Return
3040 Proc end
```

where the checking of the CPU's health is delayed 10 seconds from system startup before being checked every second. The use of **UTILITY_INIT** in this example is due to the use of **TIMER4** for the delay timer. **TIMER4** is also used by the **UTILITY.PGM** package for control of the repeat rate of the check of the CPU's health. As a result the, interrupt linkage for the **UTILITY.PGM** package must be re-established before beginning the **CHECK_CPU_HEALTH** procedure.

Section 6: Loading and Storing PCM Data Files Using TERMF

This section contains information on using TERMF to load data files from a personal computer (PC) to the PCM, store data files from the PCM to a PC, show a list of files stored in the PCM, and delete PCM files.

Note

Your PCM must have firmware version 2.50 or greater to use the techniques described in this section. If your PCM firmware is version 2.04 or lower, you should use PCOP to transfer files.

The MegaBasic **LOAD** and **STORE** commands work only with MegaBasic program files. If you need to load or store data files or binary files containing MegaBasic functions and procedures (such as the **BITFUNCS.BIN** and **BYTESWAP.BIN** files distributed with TERMF and PCOP), some other method must be used. There are two choices; you can use either TERMF or PCOP.

Caution

Using the MegaBasic **LOAD command to load files which are not MegaBasic programs to the PCM will cause unexpected results. The MegaBasic **LOAD** command modifies text files as they are loaded.**

PCOP provides menu-driven utility functions for loading and storing PCM files. If you are uncomfortable with command line computer interfaces (such as the MS-DOS command line), then PCOP is probably a better choice. Refer to the *Series 90 PCM Development Software (PCOP) User's Manual*, GFK-0487, for details on using PCOP to load and store PCM files. In order to transfer files with TERMF, you must exit from MegaBasic. You can do this from the "Ready" prompt by typing the **BYE** command. You may need to initiate a hard reset to get to the "Ready" prompt.

After you exit from MegaBasic, you may not see a prompt on your screen until you press the Enter key. Pressing the Enter key repeatedly will display a ">" prompt on the same line each time you press it. This prompt is from the PCM command interpreter. The interpreter is in its PCOP mode and does not echo the keys you type to the screen. You can switch to the interpreter's interactive (DEBUG) mode by typing two exclamation points (**!!**) and then pressing the Enter key. For Release 3.00 or later, the following message will appear:

```
INTERACTIVE MODE ENTERED
type '?' for a list of commands
```

If your PCM firmware is a version lower than 3.00, you will see "DEBUG" rather than "INTERACTIVE."

The full set of interactive mode commands is described in appendix C, *PCM Commands*. The commands used to load, store, delete, and show a list of files in the PCM RAM Disk are described in the remainder of this section.

L (Load)

Format: L <pc_filename> [<pcm_filename>]

This command directs the PCM to load the file specified by <pc_filename> to <pcm_filename> in the PCM RAM Disk. If the optional <pcm_filename> is omitted, the PC file name will be used, but without any device or file path prefix. If the file already exists in the PCM, it will be overwritten. Possible errors are:

```
File not found.
Illegal file type.
Insufficient memory.
```

The PC file name may begin with a device name. If there is no device name, the default device PC: is used to load the file. A PC disk drive and file path specification may be included in <pc_filename>, as shown for the MegaBasic **LOAD** command in chapter 4, *MegaBasic*. However, if a PC disk drive is specified, the PC: device must also be explicitly specified; for example: **L PC:A:MYFILE.BIN**. The file name is not case sensitive.

S (Save)

Format: S <pcm_filename> [<pc_filename>]

The **S (Save)** command causes a file named <pcm_filename> in the PCM RAM Disk to be saved to a PC file. If the optional <pc_filename> is omitted, the PCM filename will be used and the file will be saved to the current directory on the current disk drive. If the file does not already exist on the PC, it is created; otherwise the existing PC file will be overwritten. The <pc_filename> may include a PC disk drive and/or file path, as described above for the **L (Load)** command.

D (file Directory)

Format: D

This command prints the names of the files in the PCM RAM Disk. It returns no errors.

X (eXterminate file)

Format: X <file_name>

This command deletes a file named <file_name> in the PCM RAM Disk. An error is reported if the file is not found or is in use.

Caution

The specified file is deleted immediately. There is no confirmation prompt, nor is there any method for recovering a deleted file.

Note

See appendix C, *PCM Commands*, for additional PCM commands.

Section 7: Serial Port Setup with IOCTL and PORT_CTL.BIN

This section contains information on using the MegaBasic **IOCTL** statement to change the existing serial port configuration. There is also information on using the procedures and functions in **PORT_CTL.BIN** to send and receive serial BREAKs and to use the serial port modem control and status signals.

Serial Port Setup with IOCTL

The MegaBasic **IOCTL** statement can be used to change the Logicmaster 90 or PCOP configuration of the two PCM serial ports during program execution. **IOCTL** can change the port settings for data rate, parity, number of data bits per character, number of stop bits, flow control type and physical protocol.

Note

The **IOCTL** statement is not available in PCM firmware version 2.04 or earlier.

The format of the **IOCTL** statement is:

```
IOCTL <channel_number>, <iocctl_string>
```

Before the port can be configured, a channel for the port must be opened. The **<channel_number>** argument is the channel number which was specified when the port was opened.

The **<iocctl_string>** argument sets the port characteristics for both transmit and receive. The string must be in double quotes. The format of an **IOCTL** string is:

BAUD, PARITY, DATABITS, STOPBITS, FLOWCTL, PHYSICAL, DUPLEXMODE, TURNOFFDELAY, TABUFSIZE

| Parameter | Values | Description |
|-----------|--|---|
| BAUD | 300, 600, 1200, 2400, 4800, 9600, 19200*, or 38400 | Specifies the number of bits per second. Note that 38,400 baud is supported only by the Series 90-70 PCM, and only for RS-422 or RS-485 port configurations. |
| PARITY | O, E, N* | Specifies the type of parity checking: Odd, Even, or None. |
| DATABITS | 7 or 8* | Specifies the number of data bits per character. Use 8 unless text with 7 bit characters will be the <i>only</i> data transferred. |
| STOPBITS | 1* or 2 | Specifies the number of stop bits per character. The normal selection for 300 baud and higher is 1. |
| FLOWCTL | H*, S, or N | Specifies the flow control method: Hardware (CTS/RTS), Software (X-ON, X-OFF) or None. With Hardware flow control, RTS is turned on when the port is ready to transmit. Then, transmission begins when CTS becomes active. RTS remains on until the TURNOFFDELAY expires after the last character is sent. With Software or None flow control, RTS is not turned on, and transmission begins immediately. |

* Default selection.

| Parameter | Values | Description |
|---------------|------------------------------|--|
| PHYSICAL | 232*, 422, or 485 | Specifies the physical connection protocol for the port: RS-232, RS-422, or RS-485. RS-422 is equivalent to RS-485. All Series 90-30 PCMs support RS-232 only on COM1. IC693PCM300 supports RS-422/485 only on COM2. Invalid selections are ignored. |
| DUPLEX-MODE | 2, 4*, or p | <p>Specifies the type of physical connection:</p> <ul style="list-style-type: none"> 2 = half duplex (2 wire for RS-422/485), 4 = full duplex (4 wire for RS-422/485), p = point-to-point. <p>Available in PCM firmware version 3.00 or later.</p> <p>In point-to-point (p) mode:</p> <ul style="list-style-type: none"> The receiver for the specified port is always enabled. When PHYSICAL = 422 or 485, all RS-485 line drivers for the specified port are enabled when the command is executed and remain on continuously. <p>In fullduplex (4) mode:</p> <ul style="list-style-type: none"> The receiver for the specified port is always enabled. When PHYSICAL = 422 or 485, the RS-485 line drivers for RTS and transmitted data outputs on the specified port are turned on immediately before transmitting and remain on until TURNOFFDELAY expires after the last character is sent. At all other times, these drivers are in their high-impedance state (tri-stated). <p>In halfduplex (2) mode:</p> <ul style="list-style-type: none"> The receiver for the specified port is disabled immediately before transmitting and remains off until TURNOFFDELAY expires after the last character is sent. When PHYSICAL = 422 or 485, the RS-485 line drivers for RTS and transmitted data outputs on the specified port are turned on immediately before transmitting and remain on until TURNOFFDELAY expires after the last character is sent. At all other times, these drivers are in their high-impedance state (tri-stated). |
| TURNOFF-DELAY | 0 - 65534 Default = 0. | Specifies the time in milliseconds between the end of the last outgoing character and the time RTS is turned off (if applicable), RS-485 line drivers are tri-stated (if applicable), the receiver is enabled in half duplex mode (if applicable), and WAIT mode output statements complete execution. Available in PCM firmware version 3.00 or later. |
| TABUFSIZE | 64 - 32750 Default = 320. | Specifies the typeahead buffer size in characters for the port. The port can accept up to one less than this number of characters without overflow before an application reads the port. When overflow occurs, any additional characters will be lost. Any size in the range 64 - 32750 bytes may be specified, but the maximum may be limited by available system memory. Available in PCM firmware version 3.00 or later. |

* Default selection.

Any of the **IOCTL** parameters may be omitted. If omitted, the current value for the parameter is used. When a parameter is omitted from the middle of the string, the surrounding commas must remain to mark its place. For example, these statements:

```
Open #5, "COM2:"
Ioctl #5, "9600,,,H"
```

set the COM2 port to 9600 baud and HARDWARE flow control with whatever other parameters are currently assigned. This statement restores the PCM default settings to the port opened as channel 5:

```
Ioctl #5, "19200,N,8,1,H,232,4,0,320"
```

The default physical protocol is RS-232, except for the IC693PCM300 module, which can have RS-485 only on port 2. Also, note that the physical protocol setting, RS-232, for port 1 on the IC693PCM300, IC693PCM301, and IC693PCM311 modules cannot be changed.

Serial Port Control Using PORT_CTL.BIN

The file **PORT_CTL.BIN** is distributed with TERMF and PCOP, and is built into PCM firmware version 2.51 or later. It provides functions and procedures for sending and detecting serial breaks, for controlling the output state of serial port modem control signals, and for detecting the state of modem status signals.

If your PCM has firmware version 2.50 or earlier, you must load **PORT_CTL.BIN** to the PCM RAM disk before you can use any of its MegaBasic statements or functions. If you have version 2.50, you may use TERMF, as described in chapter 5, section 6, *Loading and Storing PCM Data Files Using TERMF*. For earlier versions, you must use PCOP.

Your MegaBasic program must use the **ACCESS** statement to make **PORT_CTL.BIN** available to your program. For firmware versions 2.50 and earlier, program:

```
xxx Access "RAM:PORT_CTL.BIN"
```

For firmware versions 2.51 and later, program:

```
xxx Access "PORT_CTL.BIN"
```

PORT_CTL.BIN provides two procedures for sending a serial break: **BREAK_ON** and **BREAK_OFF**. Their format is:

```
BREAK_ON <port_number>
BREAK_OFF <port_number>
```

The **<port_number>** argument specifies the serial port, 1 or 2, where the break is to be sent. **BREAK_ON** sets the Transmit Data line of the specified port to its logic 0 (space) state, while **BREAK_OFF** sets it to the logic 1 (mark) state.

A MegaBasic timer may be used to control the time duration of the break. For information on MegaBasic timers, see chapter 5, section 8, *Timers and Logical Interrupts*.

The **CHECK_BREAK()** and **BREAK_STATUS()** functions may be used detect a received break. Their formats are:

```
brk_rcvd% = CHECK_BREAK( <port_number> )
brk_status% = BREAK_STATUS( <port_number> )
```

The **<port_number>** argument specifies the serial port, 1 or 2, where the break is to be detected. When **CHECK_BREAK()** is called, it returns 1 if a break has been detected since the last time it was called (or the start of program execution on the first call). **BREAK_STATUS()** returns 1 when break is active at the time it is called and 0 when break is not active.

The PCM serial ports detect a break when 12 or more consecutive zero (or SPACE) bits are received on the Receive Data input. The break condition goes away when the next 1 (or MARK) bit is received.

The **RTS_ON** and **RTS_OFF** procedures are used to turn the Request To Send (RTS) output of the serial ports on and off, respectively, while **DTR_ON** and **DTR_OFF** perform the same operations for the Data Terminal Ready (DTR) output. Their formats are:

```
RTS_ON <port_number>
RTS_OFF <port_number>
DTR_ON <port_number>
DTR_OFF <port_number>
```

The **<port_number>** argument specifies the serial port, 1 or 2, where the RTS or DTR output is to be turned on or off.

The **CTS_STATUS()** and **DCD_STATUS()** functions may be used to detect the status of the Clear To Send (CTS) and Data Carrier Detect (DCD) inputs, respectively. Note that DCD is also known as Receive Line Signal Detect (RLSD) and Carrier Detect (CD).

Their formats are:

```
cts_state% = CTS_STATUS( <port_number> )
dcd_state% = DCD_STATUS( <port_number> )
```

Again, the **<port_number>** argument specifies the serial port, 1 or 2, where the status is to be tested. These functions return 1 when the corresponding status input is TRUE and 0 when FALSE.

Note that when a serial port is configured as RS-485, the **DCD_STATUS** function always returns TRUE.

Note

If the serial port is configured for hardware flow control, using RTS and DTR ON/OFF procedures is not recommended.

These functions and procedures are available in PCM firmware version 3.00 or later:

1. **ALL_SENT_STATUS (<port_number>)** returns 1 if the port has no more characters to send and 0 if the port is still sending characters.
2. **IN_LENGTH (<port_number>)** returns the number of characters which are in the type-ahead buffer of the selected port and are waiting to be processed.
3. **FLUSH_PORT <port_number>** empties the type-ahead buffer of the selected port.

This procedure, available in PCM firmware 4.03 or later, permits masking any combination of parity, overrun and framing errors on either serial port. Because serial port errors cause MegaBasic program termination unless the program explicitly handles them, you may prefer to mask these errors.

```
MASK_PORT_ERRORS <port_number>, <mask>
```

The **<port_number>** parameter specifies the serial port, 1 or 2, where errors will be masked. The **<mask>** parameter contains a set of bits to specify the errors that will be masked:

```
0010H - Parity error mask
0020H - Overrun error mask
0040H - Framing error mask
```

For example, 0010H masks parity errors only, 0050H masks both parity and framing errors, 0070H masks all three.

Section 8: Timers and Logical Interrupts

Using the interrupt handling capabilities in MegaBasic can result in programs with superior response time to external events.

Note

Logical interrupt handling in MegaBasic is described in chapter 9, *Advanced MegaBasic Programming Features*, of the *MegaBasic Programming Language Reference Manual*, GFK-0256. Because the interface between MegaBasic and the PCM operating system relies heavily on MegaBasic logical interrupts, it is important that you read this chapter before using any MegaBasic interrupt instructions.

The MegaBasic language supports 32 software interrupts. Of these, 8 are reserved for the PCM operating system, 16 for user programmable timers, 2 to indicate activity on the PCM MegaBasic program's standard input and output channels, 1 for communication timeouts, 2 for asynchronous reads and writes, and 2 for backplane communication. The remaining interrupt may be defined by your application program.

The linkage between the MegaBasic interrupt structures and the PCM hardware is done automatically during the MegaBasic initialization.

Note

Do not execute an `INTERRUPT = <interrupt number>` assignment, as you would with other versions of MegaBasic.

The discussion of foreground and background processing under MS-DOS does not apply to MegaBasic on the PCM.

At any point in a MegaBasic program, an interrupt can be defined with the following statement:

```
INTERRUPT <interrupt name>, <procedure name>, [priority]
```

This statement associates one of the PCM interrupts, specified by an interrupt name, with a user procedure, which may be either a MegaBasic or an assembly language procedure. The priority argument, if included, can be used to override the default priority of the PCM interrupt. However, do not assign priorities above 23 to the user program. Priorities above 23 are required internally to interface MegaBasic to the operating system, and the PCM will act erratically if these are used.

The following PCM interrupts and their associated default priorities are available:

Table 5-9. PCM Interrupts and Associated Defaults

| Interrupt Name | Default Priority |
|-----------------------|-------------------------|
| BKP_MSG | 23 |
| BKP_XFER | 22 |
| TIMER16 | 21 |
| TIMER15 | 20 |
| TIMER14 | 19 |
| TIMER13 | 18 |
| TIMER12 | 17 |
| TIMER11 | 16 |
| TIMER10 | 15 |
| TIMER9 | 14 |
| TIMER8 | 13 |
| TIMER7 | 12 |
| TIMER6 | 11 |
| TIMER5 | 10 |
| TIMER4 | 9 |
| TIMER3 | 8 |
| TIMER2 | 7 |
| TIMER1 | 6 |
| NOWAIT_RD | 5 |
| NOWAIT_WR | 4 |
| STD_IN | 3 |
| STD_OUT | 2 |
| COM_TIMEOUT | 1 |
| USER | 0 |

Any of the interrupts listed in this table can be assigned a priority from 0 to 23 to override the default priority. Priorities do not have to be unique; the highest priority for a PCM MegaBasic program is 23.

Caution

Although MegaBasic permits the assignment of priorities above 23, they must not be used in any program that runs on the PCM. These interrupts are used to interface MegaBasic to the PCM operating system. Any attempt to define or use these interrupts will result in erratic behavior of the program.

Once an interrupt is associated with a procedure and priority, it is still necessary to enable the interrupt. This is done with the statement:

```
INTERRUPT <interrupt name>, ON
```

At any point during program execution, an interrupt can be disabled with the statement:

```
INTERRUPT <interrupt name>, STOP
```


If the `<interrupt name>` is omitted from either of these instructions, the `ON` or `STOP` operation is applied to all of the defined interrupts.

Caution

Disabling all interrupts also disables interaction between MegaBasic and the PCM operating system. Therefore, this operation must be used with care. A PCM MegaBasic program may not be able to recover from fault conditions if interrupts are disabled when the fault occurs.

Once an interrupt has been defined, it can be redefined with a new procedure name and/or priority; but first, its old definition must be erased with this statement:

```
INTERRUPT <interrupt name>, END
```

Caution

Like the `ON` and `STOP` arguments, if the interrupt name is omitted, the `END` argument applies to all interrupt definitions. Although MegaBasic permits this, it should never be executed on a program that runs on a PCM.

Timer Interrupts

A MegaBasic program running on the PCM has 16 available timers, in addition to the timers that are automatically associated with many of the data movement routines. These 16 timers are associated with interrupts, so that a timeout on any of the timers will cause an interrupt to be posted.

The 16 timers are manipulated from a MegaBasic program with the statements:

```
Timer <timer number>, TSTART, <timer count>, [REP]
Timer <timer number>, TSTOP
```

| Parameter | Description |
|-----------------|---|
| TimerNumber | A number from 1 to 16, corresponding to timer 1 through timer 16. |
| TSTART or TSTOP | TSTART starts the timer counting from zero. TSTOP stops the timer. |
| Timer Count | The maximum count in milliseconds. This is a number from 1 to 86,400,000, the number of milliseconds in a day. |
| REP | <p>This qualifier specifies that the timer TSTART operation is to repeat. Thus, an interrupt from the specified timer is generated repetitively. If the timer count argument is a relative number (e.g., REP] is included), the interrupt occurs at the specified interval.</p> <p>For example, the statement “Timer 1, TSTART, 1000, REP” would generate an interrupt from timer 1 every second, beginning one second from the time that the statement is executed. Note that the timer 1 interrupt must be defined and enabled before the program can recognize it.</p> |

The timer interrupt procedure is specified by these statements:

```
10 Interrupt TIMER1, end
20 Interrupt TIMER1, TIMER_INT_PROC
30 Interrupt TIMER1, on
```

where **TIMER_INT_PROC** is assumed to be the name of the interrupt procedure to process the timeout.

Although the timer utilities have resolutions of one millisecond, you must be careful when using timers with small values, especially when many timers are used. Some of the more powerful MegaBasic instructions can take a millisecond or more to execute. This can potentially cause interrupts to be missed. In cases where this may be a problem, the **INTERRUPT** function can be used to determine if interrupts are being missed. Programs can be debugged using this function, which can later be edited out for increased speed.

The function call **INTERRUPT(3)** returns zero when a timer interrupt has been posted but has not been serviced. The function call **INTERRUPT(2)** returns the timer number. For more information on the **INTERRUPT** function, refer to the *MegaBasic Programming Language Reference Manual*, GFK-0256.

Backplane Interrupts

Two interrupts are provided by the PCM for backplane communication:

| Interrupt | Description |
|-----------|---|
| BKP_XFER | The BKP_XFER interrupt (number 22) is executed whenever NOWAIT mode SYSREAD or SYSWRITE data transfers between the PLC and PCM are completed. |
| BKP_MSG | The BKP_MSG interrupt (number 23) is used when a message is received from elsewhere in the Series 90 system. This may be the result of a COMMREQ block sent by the PLC CPU ladder program or a message from another PCM. |

The **BKP_XFER** interrupt occurs whenever the PCM and CPU complete a NOWAIT data exchange, either through **SYSREAD** or **SYSWRITE**. If no user procedure has been supplied for this interrupt, all backplane transfers proceed transparently. The only indication that data is being moved is the changing values of variables and the **SYSTATUS\$** function.

If NOWAIT backplane transfers are to be handled by a user interrupt procedure, the default interrupt definition must be erased and a new definition activated by the user program, as shown below.

```
10 Interrupt BKP_XFER, end
20 Interrupt BKP_XFER, USER_BKP_XFER_PROC
30 Interrupt BKP_XFER, on
```

where **USER_BKP_XFER_PROC** is the name of the user procedure to handle the backplane interrupt processing. The interrupt must be redefined before any NOWAIT backplane transfers occur.

The **BKP_MSG** interrupt occurs when data is received from transfers other than **SYSREAD** and **SYSWRITE** with the PLC.

In some applications, it is possible that, when the **BKP_MSG** interrupt is installed by the MegaBasic program, incoming messages have already arrived and are waiting to be processed. The backplane interface allows up to sixteen incoming messages to be pending. Additional incoming messages are rejected by the PCM and logged as faults in the PLC fault table.

To process any incoming messages that may have arrived before the **BKP_MSG** interrupt was installed, the MegaBasic program should call the **BKP_MSG** interrupt procedure as soon as it is installed and any other necessary data structures are initialized. Alternatively, the program could simply flush out the old messages, as shown in the following message:

```
10 Interrupt BKP_MSG, end
20 Interrupt BKP_MSG, USER_INT_PROC
30 Interrupt BKP_MSG, on
40 Repeat
50   PROCESS_MESSAGE MSG_HEADER$, MSG_STRING$
60   If MSG_HEADER = "" Then exit
70 Next
```

When the **BKP_XFER** interrupt occurs, any data read from the PLC is in a temporary buffer and has not been copied into its associated MegaBasic variable. In addition, the status information for the variable has not yet been updated. To complete the transfer, the **BKP_XFER** interrupt procedure must use the **PROCESS_XFER** statement.

PROCESS_XFER <pointer>, <handle>

| Argument | Description |
|----------|--|
| Pointer | A MegaBasic integer variable which, when PROCESS_XFER completes, contains a pointer to the variable that was transferred. Additional information can be found by passing the pointer to the SYSTATUS\$ function. For more information on pointers, refer to chapter 9 in the <i>MegaBasic Programming Language Reference Manual</i> , GFK-0256. |
| Handle | A MegaBasic integer variable which, when PROCESS_XFER completes, contains the handle specified for the variable that was transferred when it was SYSLINKed . If no handle was specified, zero is returned. The handle can be used for any purpose by the application. Typically, it is used to organize variables into groups that receive similar processing. |

Note

Both the pointer and handle arguments are MegaBasic variables whose values are set by **PROCESS_XFER** for later use in the interrupt service routine. Since they are set by **PROCESS_XFER**, they do not have to be initialized before the **PROCESS_XFER** statement is executed.

The interface between the PCM and PLC allows several backplane transfers to complete at once. In order to ensure that all transfers are processed, the **BKP_XFER** interrupt procedure must use the **PROCESS_XFER** statement repeatedly until zero is returned in the pointer variable, indicating that there are no more unprocessed transfers. In the following example, the **BKP_XFER** interrupt procedure does not return until the pointer variable is set to zero by **PROCESS_XFER**.

```
500 Def Proc USER_BKP_XFER_PROC
510 Local POINTER%, HANDLE%
520 Repeat
530   PROCESS_XFER POINTER%, HANDLE%
540   If POINTER% = 0 Then Return
550   Rem Process the data, use POINTER% and
560   Rem HANDLE% to find out what was just transferred
570 Next
580 Proc end
```

If a user procedure is supplied for the **BKP_XFER** or **BKP_MSG** interrupt, the previous interrupt definition must be erased and a new definition activated by the user program before the new procedure is used, as shown above and in the following example.

Example using NOWAIT SYSREAD Commands

In this example, a MegaBasic program monitors four analog input channels. The data from each channel is read and scaled by the PCM, checked for overrange, and displayed as a bar graph on a screen.

```
channel 1   XXXXXXXXX
channel 2   XXXXXXXXXXXXXXXXXXXX
channel 3   XXXXXXXXXXXXXXXXXXXXXXXX
channel 4   XXXXXXXXXXXXX
          +-----+-----+-----+-----+
```

There are several ways to solve this problem. If the analog inputs were located together in the PLC and the MegaBasic program did not have to do any other processing, the program could wait in a loop, reading the inputs and displaying them as follows:

```
10 Dim integer INPUT_TBL(3)
20 SYSLINK INPUT_TBL, "%AI001", INT16
30 Do
40   SYSREAD INPUT_TBL
50   For I = 0 to 3
60     SCALE INPUT_TBL(I)
70     DISPLAY INPUT_TBL(I)
80     If INPUT_TBL(I) > OVERRANGE Then [
90       SEND_ALARM I
100   ]
110 Next
120 Next
```

Note

It is assumed that the **OVERRANGE** constant and the **SCALE**, **DISPLAY**, and **SEND_ALARM** procedures are defined elsewhere in the program or in another package.

If the four input channels are not located in contiguous PLC memory and the MegaBasic program has additional processing to do, so that it cannot stay in a dedicated loop, the inputs can be processed more efficiently with an interrupt procedure as follows:

```

10 Def integer INPUT_TBL0
20 Def integer INPUT_TBL1
30 Def integer INPUT_TBL2
40 Def integer INPUT_TBL3
50 SYSLINK INPUT_TBL0, "%AI001", INT16, 0
60 SYSLINK INPUT_TBL1, "%AI025", INT16, 1
70 SYSLINK INPUT_TBL2, "%AI079", INT16, 2
80 SYSLINK INPUT_TBL3, "%AI133", INT16, 3
90 SYSREAD INPUT_TBL0, NOWAIT, 100
100 SYSREAD INPUT_TBL1, NOWAIT, 100
110 SYSREAD INPUT_TBL2, NOWAIT, 100
120 SYSREAD INPUT_TBL3, NOWAIT, 100
130 Interrupt BKP_XFER, end
140 Interrupt BKP_XFER, PROCESS_INPUTS
150 Interrupt on
    ...
    ...
500 Def Proc PROCESS_INPUTS
510 Local DATA_PTR%, CHANNEL_NUM%
520 Repeat
530   PROCESS_XFER DATA_PTR%, CHANNEL_NUM%
540   If DATA_PTR% = 0 Then [
550     Return
560   ]
570   SCALE *DATA_PTR%
580   DISPLAY *DATA_PTR%, CHANNEL_NUM%
590   If *DATA_PTR% > OVERRANGE Then [
600     SEND_ALARM CHANNEL_NUM%
610   ]
620 Next
630 Proc end

```

The **BKP_XFER** interrupt and the **PROCESS_XFER** statement are used here because the data messages are transferred from PLC memories to the PCM. **BKP_MSG** and **PROCESS_MESSAGE** would be used if the PLC sent this data via a COMMREQ.

This second solution is more complex than the first because four separate variables are defined, linked and transferred. However, they can all be processed with the same interrupt procedure. Also, since each variable is transferred only once every 100 milliseconds, the MegaBasic program can spend most of its time on other processing. In either case, the **SCALE**, **DISPLAY** and **SEND_ALARM** procedures are identical to the procedures in the previous example and are called with the same arguments.

Section 9: COMMREQs and Other Backplane Messages

The previous section describes processing of data sent by the PLC CPU in response to **SYSREAD** and **SYSWRITE** statements. It is also possible for MegaBasic programs to receive unexpected messages from the CPU or elsewhere in the PLC. These messages can be messages from COMMREQ function blocks in the PLC program, responses to request messages sent to the CPU (for example, by the **GENERIC.PGM** package), or messages from another PCM. Since they reach the PCM on the PLC backplane, they are referred to in this section as **backplanemessages**.

PROCESS_MESSAGE Statement

When backplane messages arrive, the PCM stores them in a temporary holding queue. MegaBasic programs must use the **PROCESS_MESSAGE** statement to move them into MegaBasic variables. The format of the **PROCESS_MESSAGE** statement is:

```
PROCESS_MESSAGE <header string> [<message string>]
```

The **PROCESS_MESSAGE** arguments are described in the following table.

| Field | Description |
|------------------|--|
| <header string> | This parameter must contain the name of a string variable which has been dimensioned to at least 32 bytes. Executing the PROCESS_MESSAGE statement puts the message header into it. Tables 5-9 and 5-10 describe the header format. |
| <message string> | This optional parameter contains the name of a string variable which has been dimensioned to at least 256 bytes. After the PROCESS_MESSAGE statement is executed, it may or may not contain data. The current length indicates whether there is data and, if so, how much. The actual data format depends on the application. |

Note

If the optional message string parameter is omitted and the message included a message string, the data in the message string will be lost.

After the **PROCESS_MESSAGE** statement is executed, the header string variable will contain this data:

Table 5-10. Structure of Backplane Messages

| | | |
|----|---------------------|---------------|
| 0 | Status | 2 bytes |
| 2 | Reserved | 3 bytes |
| 5 | Priority | 1 byte |
| 6 | Application Defined | 1 byte |
| 7 | Message Type | 1 byte |
| 8 | Source | 4 bytes |
| 12 | Destination | 4 bytes |
| 16 | Reserved | 0 or 8 bytes |
| 32 | Application Defined | 16 or 8 bytes |

The parts of backplane messages are explained in the following table:

Table 5-11. Backplane Message Fields

| Field | Description |
|--------------|---|
| Status | This field is zero when there are no errors. All non-zero status values are error values from the operating system. |
| Priority | Specifies whether the message was sent high or low priority. Bit 7 is set to 1 to indicate high priority. As a rule of thumb, all messages that do not require single-sweep response time should be sent low priority. However, high priority does not, in itself, guarantee that transfers complete in one PLC sweep. |
| Message Type | Used by the receiving task to aid in decoding the message. The message type depends on the source of the message. See "Identifying the Source of Backplane Messages," below. |
| Source | Identifies the sender of the message. The two upper bytes, starting at offset 10, are unused. The two lower bytes are formatted as follows: <div style="display: flex; justify-content: space-around; align-items: flex-end;"> <div style="text-align: center;"> <div>16 10</div> <div style="border: 1px solid black; padding: 2px;">Task Id #</div> <div>7 bits</div> </div> <div style="text-align: center;"> <div>9 5</div> <div style="border: 1px solid black; padding: 2px;">Slot #</div> <div>5 bits</div> </div> <div style="text-align: center;"> <div>4 1</div> <div style="border: 1px solid black; padding: 2px;">Rack #</div> <div>4 bits</div> </div> </div> <p>Rack and slot numbers are used to identify a given module in the Series 90 PLC system. ID identifies a particular program or software process on the module. For messages to other PCM tasks, IDs 1 and 2 are used by CCM on ports 1 and 2, respectively, and ID 3 is used by MegaBasic.</p> |
| Destination | Identifies the destination of the message. When MegaBasic receives it, the ID value is 3, and the slot and rack values correspond to the location of the PCM. |
| Reserved | The data in these fields varies. |

Using the BKP_MSG Interrupt

COMMREQ messages and messages from another smart module are sometimes referred to as ***unsolicited communication*** because they are not requested by the PCM. These messages arrive at unexpected times. However, the arrival time of all backplane messages is somewhat unpredictable. After a generic request message is sent to the PLC CPU, several MegaBasic statements are usually executed before the response message arrives.

The most efficient way to handle unpredictable messages is with the **BKP_MSG** interrupt. This program fragment shows how to use it.

```
100 Dim MESSAGE_HDR$(32)
110 Dim MESSAGE_TXT$(256)
120 Def proc USER_BKP_MSG_PROC
130 Repeat
140   PROCESS_MESSAGE MESSAGE_HDR$, MESSAGE_TXT$
150   If MESSAGE_HDR$ = "" then Return
160   Rem process the data here
170 Next
180 Proc end

. . .

500 Interrupt BKP_MSG, end
510 Interrupt BKP_MSG, USER_BKP_MSG_PROC
520 Interrupt BKP_MSG, on
```


Interpreting COMMREQ Messages

Execution of a COMMREQ function block in the PLC CPU will cause one of these four message types to be sent to the target PCM. All COMMREQ messages contain rack 0, slot 1, ID 9 in the source field.

The following table explains how these four message types differ:

| Type | Description |
|------------|--|
| 130 or 131 | <p>The body of the COMMREQ message is contained in a message string that is copied to the string provided as the second argument to PROCESS_MESSAGE. The word at offset 18 in the header specifies the length of the text string in bytes. The length can also be found by calling the LEN function with the message string as an argument.</p> <p>A message type of 131 indicates that the PLC CPU is waiting for the completion of message processing. This message type are used when the WAIT option is specified in the COMMREQ command block. This option is not recommended, however, because execution of the PLC program will not continue until the MegaBasic program sends a status message back to the CPU. When multiple tasks are active in the PCM, a significant amount of time may elapse before the MegaBasic program can do so.</p> |
| 194 | <p>The body of the COMMREQ message is contained in the last 12 bytes of the messageheader. No information is provided on the length of the message, unless it is built into the message itself.</p> |
| 195 | <p>The body of the COMMREQ message is contained in the last 8 bytes of the message header. No information is provided on the length of the message, unless it is built into the message itself.</p> <p>A message type of 195 indicates that the PLC CPU is waiting for the completion of message processing. This message type are used when the WAIT option is specified in the COMMREQ command block. This option is not recommended, however, because execution of the PLC program will not continue until the MegaBasic program sends a status message back to the CPU. When multiple tasks are active in the PCM, a significant amount of time may elapse before the MegaBasic program can do so.</p> |

This MegaBasic program fragment shows how to extract data from a COMMREQ message.

```

100 Dim MESSAGE_HDR$(32)
110 Dim MESSAGE_TXT$(256)
120 Dim COMMREQ_DATA$(256)

. . .

500 PROCESS_MESSAGE MESSAGE_HDR$, MESSAGE_TXT$
510 MSG_TYPE% = asc(MESSAGE_HDR$(8))
520 If MSG_TYPE% = 194 then [
530   COMMREQ_DATA_SIZE% = 12
540   COMMREQ_DATA$ = MESSAGE_HDR$(21:12)
550 ] Else If MSG_TYPE% = 195 then [
560   COMMREQ_DATA_SIZE% = 8
570   COMMREQ_DATA$ = MESSAGE_HDR$(25:8)
580 ] Else If MSG_TYPE% = 130 or MSG_TYPE% = 131 [
590   COMMREQ_DATA_SIZE% = len(MESSAGE_TXT$)
600   COMMREQ_DATA$ = MESSAGE_TXT$(1:COMMREQ_DATA_SIZE%)
610 ] Else [
620 Rem not a COMMREQ - do something else
630 ]

```

Programming the PLC COMMREQ Function Block

The PLC CPU uses the parameters of the COMMREQ function block and a command block to define the data required to communicate with the PCM. When the COMMREQ function receives power flow, a block of data is sent to the PCM. The Communications Request 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 PCM and program execution resumes immediately. This is referred to as **NOWAIT mode**.

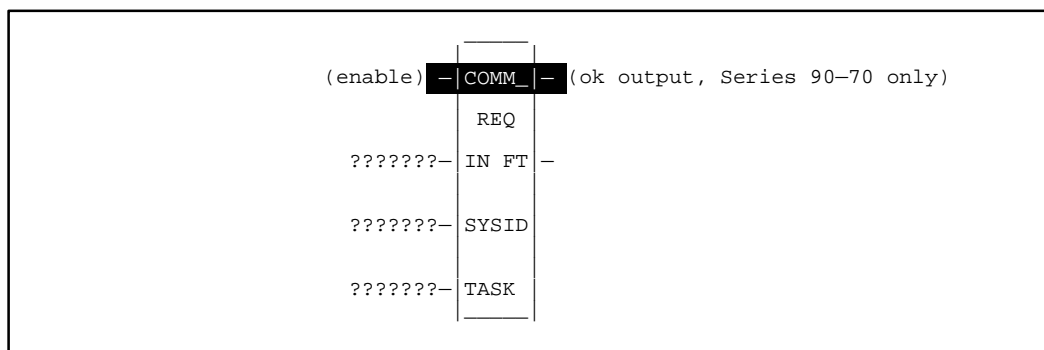
If the command block specifies that the program waits for a reply, the command block contents are sent to the PCM and the CPU waits for a reply. The maximum length of time the PLC will wait for the device to respond is specified in the command block. If the device does not respond in that time, program execution resumes. This is referred to as **WAIT mode**.

Caution

It is recommended that the flag be set to NOWAIT. Otherwise, the time spent by MegaBasic could negatively impact the CPU sweep.

Format of the COMMREQ Instruction

The COMMREQ instruction has four input parameters and two output parameters. When the COMMREQ function receives power flow, a command block of data is sent to the communications TASK. The command block begins at the reference specified using the parameter IN. The PCM to be communicated with is indicated by entering its rack and slot number for SYSID.



IN: Specifies the location of the command block. It may be any word-oriented user reference (%P, %L, %R, %AI, or %AQ).

SYSID: A hexadecimal value that gives the rack and slot location of the PCM to which the COMMREQ is being sent. Entries have this format:

| | | | | | | |
|------|-------|---|---|--------|-------|------|
| | | R | S | | | 0102 |
| rack | _____ | | | rack 1 | _____ | |
| slot | _____ | | | slot 2 | _____ | |

If the SYSID is incorrectly programmed for a rack and slot that does not contain a PCM, the PLC will detect the error and will not send the communications request.

Additional examples:

| Rack | Slot | Hex Word Value |
|------|------|----------------|
| 0 | 4 | 0004h |
| 3 | 4 | 0304h |
| 2 | 9 | 0209h |
| 7 | 2 | 0702h |

TASK: The following table lists the applicable task numbers for the PCM.

| Task Number | Description |
|-------------|-------------|
| 3 | MegaBasic |

If the task number programmed for the PCM is not valid, an application fault, “COMMREQ BAD TASKID,” is logged in the PLC fault table. This can occur if the task on the COMMREQ is misprogrammed, if the PCM has been hard reset so that MegaBasic is not running, or if the PCM is not configured correctly for MegaBasic.

OK and FT: The OK and FT (function faulted) output parameters can provide power flow to optional logic, which can verify successful completion of the COMMREQ. Note that the Series 90-30 COMMREQ has no OK output. OK and FT may have these states:

| ENable | Error? | OK output | FT output |
|------------|--------------|-----------|-----------|
| active | no | true | false |
| ctive | yes | false | true |
| not active | no execution | false | false |

The COMMREQ always passes power flow to the OK output in NOWAIT mode. In WAIT mode, the function passes power flow to the OK output, unless the timeout period is exceeded or a zero timeout period has been specified. The FT output may be set true if:

- The specified target address is not present.
- The specified task is not valid for the device.
- The data length is 0.
- The status pointer address (part of the command block) does not exist. This may be due to an incorrect user reference type selection, or an address that is out of range within that reference type.

If there are errors in the portion of the command block used specifically by the PCM, these errors are reflected in the value returned in the status location, not in the FT output.

MegaBasic COMMREQ Command Block

The format of the command block is shown below.

| | |
|----------------------------|--------------------------|
| Length | address(word 1) |
| Wait/NoWait Flag | address + 1 (word 2) |
| Status Pointer Memory | address + 2 (word 3) |
| Status Pointer Offset | address + 3 (word 4) |
| Idle Timeout Value | address + 4 (word 5) |
| Maximum Communication Time | address + 5 (word 6) |
| Data Block | address + 6 (word 7) |
| | t o |
| | address + 133 (word 134) |

| Parameter | Description |
|----------------------------|---|
| Length | <p>The length, in words, of the application specific data block is the first word in the command block. The data block may contain from 1 to 128 words (2 - 256 bytes) of data to be sent to a MegaBasic program on the PCM.</p> <p>The length and content of the data are determined by the user. The application could, for example, send the PCM a single word with bits set to enable certain operations on the PCM. This would be similar to the control byte in the Series Six ASCII BASIC Module, except that 16 bits are available instead of eight. Alternatively, the PLC could send a 256-byte block of data for the MegaBasic program to process.</p> |
| Wait/NoWait Flag | The Wait/NoWait flag should be set to 0 (NOWAIT mode). While 1 (WAIT mode) can be used with MegaBasic programs, NOWAIT mode is strongly recommended. |
| Status Pointer | <p>The status pointer specifies the PLC memory location where the PCM program should return status information. Unlike CCM COMMREQs, no status information is returned automatically. The PCM program must contain code for this purpose.</p> <p>The first word of the status pointer (memory type) contains a reference type selector value:</p> <ul style="list-style-type: none"> 16 = discrete input table (%I). 18 = discrete output table (%Q). 8 = register memory (%R). 10 = analog input table (%AI). 12 = analog output table (%AQ). <p>The second word (offset) contains the offset within the specified reference type to the status data. The location and content of the status information depends upon the user application.</p> |
| Idle Timeout Value | Set the value to zero (e.g., NOWAIT). |
| Maximum Communication Time | Set the value to zero (e.g., NOWAIT). |

MegaBasic COMMREQ Example

The following example is a complete MegaBasic COMMREQ application for the PCM, including PLC Ladder logic and MegaBasic program segment. For this example, the PCM must be configured using PCOP to permit LED 1 to be controlled by the MegaBasic program. The PLC program uses the COMMREQ to command the PCM to blink or turn off the PCM's User LED 1. This is done by sending a single word with the following translation in the MegaBasic program:

| | |
|---|-------------------------|
| 1 | Blink LED continuously. |
| 2 | Blink LED once. |
| 3 | Turn LED off. |
| 4 | Blink LED continuously. |

Note that it is also possible to simply turn the LED on, but this program does not use that indication.

On the first scan, the COMMREQ command block and its data block are initialized using the BLKMV function block. The command block is located in %R50-%R55. %R50, the data block length, is 1; it contains a single word command for the MegaBasic program. %R51 contains a 0 for NOWAIT mode. The status return is in %R200, since %R52 is 8 for %R memory and %R53 is 199. %R54 and %R55 are ignored in NOWAIT mode and are simply initialized to 0. The data block begins at %R56 and contains one word. In this example, the PLC ladder instructs the PCM to blink the LED continuously since %R56 is set to 4.

The program ensures that the PCM has had time to initialize before sending the first COMMREQ. This is done with a 5.0 second timer function block in rung 7. The timer count register, %R0001, was also initialized to 0 by a MOVE_INT block in rung 5. After the timer has expired, %T0001 transitions on to stop the timer, and %T0002 is on for one sweep.

When %T0002 is on, power flow is provided to rung 9. The return status location for the COMMREQ block, %R0200, is set to zero by a MOVE_INT function block. Then the COMMREQ is sent to the PCM. The IN parameter to the COMMREQ gives location %R50 for the command block. SYSID is 0002, indicating the PCM is located in rack 0, slot 2. The TASK parameter indicates that this COMMREQ is for the MegaBasic program since its value is 3.

The MegaBasic program defines two string variables, **CMRQ_HDR\$** and **CMRQ_TXT\$** for receiving the COMMREQ data block. Lines 110-140 remove any messages that may have been received by the PCM before this program was started. Applications should flush the incoming buffer in this way. Lines 150-160 keep MegaBasic from terminating the program.

Lines 80-100 install a user procedure called **USER_BKP_MSG_PROC** for the COMMREQ interrupt. The procedure processes any received messages until the message string **CMRQ_HDR\$** is NULL, indicating no more available messages. **PROCESS_MESSAGE** fills **CMRQ_HDR\$** and **CMRQ_TXT\$** with the next incoming message. Note that **CMRQ_TXT\$** is not used in this example. Also note that the **REM** statements on lines 250-270 can be uncommented to display the actual message contents to the screen.

Line 280 converts the first (and only) word in the COMMREQ data to an integer and stores the value in **LED_CMD%**. The IF statements in lines 290-330 determine the proper action to take, based on the value of **LED_CMD%**. Note that if the program needs to return a status for the COMMREQ to the PLC, a **SYSLINK** to %R200 in the main program and a line at 205 to **SYSWRITE** a value to %R200 would be added. The format and value of %R200 is totally user-definable.

02-21-92 09:07 GE FANUC SERIES 90-30/90-20 DOCUMENTATION (v3.00) Page 1
MegaBasic COMMREQ example

```
[ START OF LD PROGRAM MB_CRQ3 ]      ( *                               *)
[ VARIABLE DECLARATIONS ]
[ BLOCK DECLARATIONS ]
[ START OF PROGRAM LOGIC ]

(*****
(* Initialize the delay timer and the MegaBasic COMMREQ command block. *)
*****)

<< RUNG 5 STEP #0002 >>

FST_SCN
+---] [---| BLKMOV |-----| MOVE |---
          INT      INT

CONST - IN1 Q+---%R0050  CONST - IN Q ---%R0001
+00001 +00000 +00000  LEN
                                00001

CONST - IN2
+00000

CONST - IN3
+00008

CONST - IN4
+00199

CONST - IN5
+00000

CONST - IN6
+00000

CONST - IN7
+00004

(*****
(* Delay the COMMREQ 5.0 Sec. %T0001 is guaranteed to be off when the PLC *)
(* transitions to RUN mode. %T0002 is on for only one sweep when the timer *)
(* times out. *)
*****)
```

Program: MB_CRQ3

C:\LM90\MB_CRQ3

Block: _MAIN

```

02-21-92  09:07 GE FANUC SERIES 90-30/90-20 DOCUMENTATION (v3.00)      Page      2
MegaBasic COMMREQ example

| << RUNG 7  STEP #0006 >>
|
| %T0001                                     %T0001
| +---] / [---| TMR |-----| (S)-----|
|              | 0.10s |
|              | PV   |-----| %T0002
| CONST -      +00050 |-----| ( )-----|
|
| %R0001
|
| (*****
| (* Clear the COMMREQ return status location; then send a MegaBasic COMMREQ *)
| (* to TASK ID 3 in the PCM at rack 0, slot 2.  If there is an error,      *)
| (* %T0003 will latch.                                                       *)
| (*****
|
| << RUNG 9  STEP #0011 >>
|
| %T0002                                     %T0003
| +---] [---| MOVE |-----| COMM |-----| (S)-----|
|              | INT |
|              | IN Q+---%R0200  %R0050 - IN FT |
| CONST -      +00000 | LEN |
|              | 00001 |
|
|                      CONST - SYSID
|                      0002
|
|                      CONST - TASK
|                      00000003
|
| [          END OF PROGRAM LOGIC          ]

```

Block: MAIN

MegaBasic Blink LED Program Example

```

10 Rem *****
20 Rem * This example program controls the PCM user LED 1 based on a *
30 Rem * command value received in a COMMREQ message. The LED must be *
40 Rem * configured for use by the MegaBasic task. *
50 Rem *****
60 Dim CMRQ_HDR$(32)
70 Dim CMRQ_TXT$(32)
80 Interrupt BKP_MSG, end
90 Interrupt BKP_MSG, USER_BKP_MSG_PROC
100 Interrupt BKP_MSG, on
110 Repeat
120   PROCESS_MESSAGE CMRQ_HDR$, CMRQ_TXT$
130   If CMRQ_HDR$ = "" then Exit
140 Next
150 Repeat
160 Next

170 Def proc USER_BKP_MSG_PROC
180   Repeat
190     PROCESS_MESSAGE CMRQ_HDR$, CMRQ_TXT$
200     If CMRQ_HDR$ = "" then Return

210 Rem *****
220 Rem * Uncomment the next 3 lines to print the COMMREQ message. The *
230 Rem * message bytes are printed in decimal format. *
240 Rem *****
250 Rem For I% = 1 to 16; Print asc(CMRQ_HDR$(I%)), " "; Next I%; Print
260 Rem For I% = 17 to 32; Print asc(CMRQ_HDR$(I%)), " "; Next I%; Print
270 Rem Print
280   LED_CMD% = asc(CMRQ_HDR$(21))
290   If LED_CMD% = 1 or LED_CMD% = 4 then [
300     SET_LED 1, 4 ;Rem blink LED 1 continuously
310   ] Else If LED_CMD% = 2 or LED_CMD% = 3 then [
320     SET_LED 1, LED_CMD% ;Rem 2: blink LED 1 once
330   ] ;Rem 3: turn LED 1 off
340   Next
350   Return
360 Proc end

```

Controlling COMMREQs

The temporary queue where backplane messages are stored can hold only a limited number. When the queue overflows, the PCM is unable to receive another backplane message until **PROCESS_MESSAGE** is called. COMMREQs sent to the PCM are lost, and a fault is posted to the PLC fault table. Overflow can be prevented by regulating the rate at which the PLC CPU sends repeated COMMREQs. The next example shows how.

The PLC ladder program for this example is based on the previous one. However, it sends COMMREQs repeatedly rather than just once. As before, the first COMMREQ is delayed 5 seconds after the program runs. Then, the PLC waits for the PCM to signal that it is ready for another COMMREQ. The PCM sends its signal by using the **SYWRITE** statement to put a non-zero value into the COMMREQ status register in the PLC CPU.

Rung 9 is added to the ladder to test the COMMREQ status register, %R0200, for a non-zero value. When the PCM has sent its signal, the test succeeds, and %T0003 is turned on to permit another COMMREQ.

The first COMMREQ needs to be sent before the PCM can send a signal. A new **MOVE_INT** block in rung 5 sets the status register to one on the first scan. Then rung 9 permits the first COMMREQ.

The MegaBasic program in this example finds the status register location for each COMMREQ, prints its data, and then signals the PLC CPU to send the next one. Most of the work is done in **USER_BKP_MSG_PROC**, which begins at line 550.

The string variables used in the program are dimensioned in lines 80-110. The main program begins at line 120. In lines 150-180, it processes and discards any messages which were in the temporary queue when the program started. Then, in lines 220-240, **USER_BKP_MSG_PROC** is assigned to the MegaBasic **BKP_MSG** logical interrupt. Finally, the COMMREQ signal is initialized in line 290, and the program waits in an endless loop for a COMMREQ to be received.

When a COMMREQ message arrives, **USER_BKP_MSG_PROC** puts it into **CMRQ_HDR\$** by calling **PROCESS_MESSAGE** at line 630. This version of **USER_BKP_MSG_PROC** also processes messages in a loop, and normally exits at line 640 the second time through. When there is a message, **CMRQ_RCVD%** is set to one in line 730, and **CMRQ_TXT\$** is tested in line 740. If **CMRQ_TXT\$** contains no data, the status pointer in **CMRQ_HDR\$** starts at character position 17 in **CMRQ_HDR\$**, there are 12 data bytes or less, and the data starts at character position 21. Twelve bytes are copied to **CMRQ_DATA\$** in line 780, because the actual data size is unknown.

If there is data in **CMRQ_TXT\$**, the status pointer in **CMRQ_HDR\$** starts at character position 25. The data size is found from the current length of **CMRQ_TXT\$**, and the actual number of data bytes is copied to **CMRQ_DATA\$** in line 830.

The status pointer type is in a single byte whose position in **CMRQ_HDR\$** is stored in **T%**. It is converted from a character to a value in line 890. The status pointer offset value is a 16 bit unsigned value. MegaBasic has no built-in facility for extracting two characters from a string and converting them to an unsigned integer, so line 900 does it the crude way.

The CASE block beginning at line 910 converts any one of the status pointer types it recognizes to the corresponding string. Unrecognized types result in a NULL string, which is ignored. If the type was recognized, the offset value is converted to a string and added to the pointer type string in line 1070. The complete COMMREQ status address is printed at line 1080. This completes the processing done by `USER_BKP_MSG_PROC`.

When the `BKP_MSG` logical interrupt ends, the main program loop resumes. At line 310 it detects that a COMMREQ was received, and it resets `CMRQ_RCVD%` at line 320. Lines 370-420 print the data size and the data. The data is presented as bytes in hexadecimal format, which often makes it easier to interpret 16 bit values separated into bytes. If you prefer decimal integer format, you can remove the format specifier string, `%"3H3"`, plus its trailing comma, from the print statement in line 400.

In lines 480-510, `NEXT_CMRQ%` is `SYSLINK`d to the status register address in `STAT_PTR$`. Then a `SYSWRITE` statement is used to signal the PLC CPU, and `NEXT_CMRQ%` is `UNLINK`d. When the loop repeats, the test at line 310 fails until the next COMMREQ arrives.

The technique for regulating COMMREQs shown in this example could be used with a ladder program containing several COMMREQ function blocks. Each WAIT mode COMMREQ block must have its own status separate status register. This program responds to the appropriate status register for each COMMREQ message. The developer of the MegaBasic program does not need to know in advance what the status registers will be.

Here is a tip to make the process of debugging a program using this regulation method easier. The design of the ladder program assumes the PCM program starts before the first COMMREQ is sent and continues to respond to COMMREQs indefinitely. But MegaBasic programs are started and stopped frequently while they are being developed. When this program stops, the PLC CPU never receives a signal to send the next COMMREQ.

The ladder program can be stopped and then started at the same time as the MegaBasic program, but it is easier to leave the CPU running. After line 290 in the example, add 3 lines to `SYSLINK CMRQ_RCVD%` to `%R0200`, `SYSWRITE` it, and `UNLINK` it. The added lines will cause the ladder program to send a COMMREQ whenever the MegaBasic program starts. When the program is working correctly, these lines can be remarked out.

02-20-92 15:46 GE FANUC SERIES 90-30/90-20 DOCUMENTATION (v3.00) Page 1
Regulating MegaBasic COMMREQs

```
[ START OF LD PROGRAM REG_CRQ ]      ( *                               *)
[ VARIABLE DECLARATIONS ]
[ BLOCK DECLARATIONS ]
[ START OF PROGRAM LOGIC ]
```

```
(*****
(* Initialize the delay timer, the MegaBasic COMMREQ command block, and *)
(* its status location. *)
(*****)
```

<< RUNG 5 STEP #0002 >>

| FST_SCN | BLKMOV | INT | MOVE | INT | MOVE | INT |
|-----------------------|----------------------|----------------------|------|-----|------|-----|
| CONST - IN1 Q -%R0050 | CONST - IN Q -%R0001 | CONST - IN Q -%R0200 | | | | |
| +00010 | +00000 | LEN 00001 | | | | |
| CONST - IN2 | | | | | | |
| +00000 | | | | | | |
| CONST - IN3 | | | | | | |
| +00008 | | | | | | |
| CONST - IN4 | | | | | | |
| +00199 | | | | | | |
| CONST - IN5 | | | | | | |
| +00000 | | | | | | |
| CONST - IN6 | | | | | | |
| +00000 | | | | | | |
| CONST - IN7 | | | | | | |
| +00004 | | | | | | |

```
(*****
(* Delay the first COMMREQ 5.0 Sec. %T0001 is guaranteed to be off when *)
(* the PLC transitions to RUN mode. *)
(*****)
```

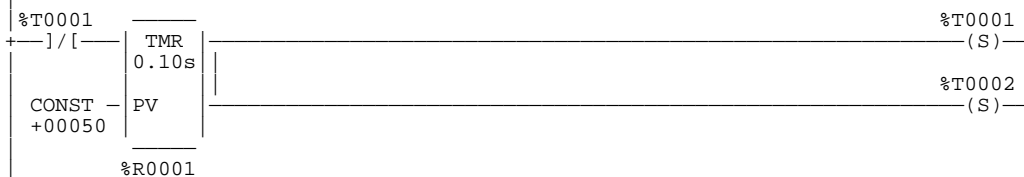
Program: REG_CRQ

C:\LM90\REG_CRQ

Block: _MAIN

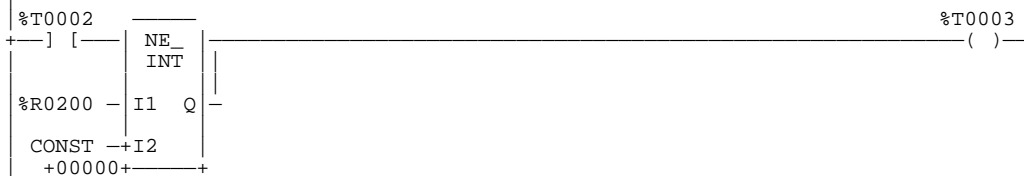
02-20-92 15:46 GE FANUC SERIES 90-30/90-20 DOCUMENTATION (v3.00) Page 2
Regulating MegaBasic COMMREQs

<< RUNG 7 STEP #0007 >>



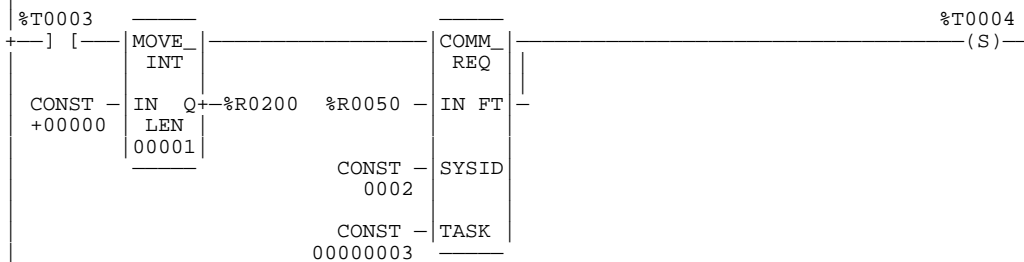
(*****
(* After the initial delay, enable the first COMMREQ. Then, wait until *)
(* the PCM acknowledges each one before enabling the next. *)
(***)

<< RUNG 9 STEP #0012 >>



(*****
(* Send a MegaBasic COMMREQ in NOWAIT mode to ID 3 in the PCM at rack 0, *)
(* slot 2, but only when the previous one is successfully acknowledged. *)
(***)

<< RUNG 11 STEP #0016 >>



[END OF PROGRAM LOGIC]

Program: REG_CRQ

C:\LM90\REG_CRQ

Block: _MAIN

```

10 Rem *****
20 Rem * This example program shows how to: *
30 Rem * 1. Determine the COMMREQ status location in PLC memory; *
40 Rem * 2. Regulate COMMREQs from the PLC program so that the PCM *
50 Rem * backplane message queue never overflows; and *
60 Rem * 3. Determine the amount of data in the COMMREQ. *
70 Rem *****
80 Dim CMRQ_HDR$(32)
90 Dim CMRQ_TXT$(256)
100 Dim CMRQ_DATA$(256)
110 Dim STAT_PTR$(8)

120 Rem *****
130 Rem * Throw away messages already in the backplane queue. *
140 Rem *****
150 Repeat
160   PROCESS_MESSAGE CMRQ_HDR$, CMRQ_TXT$
170   If CMRQ_HDR$ = "" then Exit
180 Next

190 Rem *****
200 Rem * Assign USER_BKP_MSG_PROC to the BKP_MSG interrupt. *
210 Rem *****
220 Interrupt BKP_MSG, end
230 Interrupt BKP_MSG, USER_BKP_MSG_PROC
240 Interrupt BKP_MSG, on

250 Rem *****
260 Rem * Initialize the NEXT_CMRQ% as the PLC signal. Then, wait for *
270 Rem * COMMREQs in an endless loop *
280 Rem *****
290 NEXT_CMRQ% = 1
300 Repeat
310   If CMRQ_RCVD% <> 0 then [
320     CMRQ_RCVD% = 0

330 Rem *****
340 Rem * A COMMREQ arrived. Print the data size and then the data. *
350 Rem * The data is shown as bytes in hexadecimal format. *
360 Rem *****
370   Print "COMMREQ data size =", CMRQ_DATA_SIZE%, " bytes"
380   Print "COMMREQ data:"
390   For I% = 1 to CMRQ_DATA_SIZE%
400     Print "%3H2", asc(CMRQ_DATA$(I%)),
410   Next I%
420   Print; Print

430 Rem *****
440 Rem * Signal the PLC CPU to send the next COMMREQ. NEXT_CMRQ% is *
450 Rem * SYSLINKed and UNLINKed each time through the loop; the CPU *
460 Rem * could use a different status register for each COMMREQ. *
470 Rem *****
480   If STAT_PTR$ <> "" then [
490     SYSLINK NEXT_CMRQ%, STAT_PTR$, UINT
500     SYSWRITE NEXT_CMRQ%
510     UNLINK NEXT_CMRQ%
520   ]
530 ]
540 Next

```

```

550 Rem *****
560 Rem * This procedure handles the BKP_MSG interrupt. *
570 Rem * Process messages until no more are available. *
580 Rem *****
590 Def proc USER_BKP_MSG_PROC
600   Local T% ;Rem Type Index
610   Local O% ;Rem Offset Index
620   Repeat
630     PROCESS_MESSAGE CMRQ_HDR$, CMRQ_TXT$
640     If CMRQ_HDR$ = "" then Return

650 Rem *****
660 Rem * There is a COMMREQ message. It is assumed to be a NOWAIT *
670 Rem * mode COMMREQ. Put the data size into CMRQ_DATA_SIZE%. If *
680 Rem * there are 12 data bytes or less, there is no way to know the *
690 Rem * exact data size unless it is included in the COMMREQ data *
700 Rem * itself. *
710 Rem * Next, copy the COMMREQ data to CMRQ_DATA$. *
720 Rem *****
730   CMRQ_RCVD% = 1
740   If CMRQ_TXT$ = "" then [
750     T% = 17
760     O% = 19
770     CMRQ_DATA_SIZE% = 12 ;Rem it could be less
780     CMRQ_DATA$ = CMRQ_HDR$(21:12)
790   ] Else [
800     T% = 25
810     O% = 27
820     CMRQ_DATA_SIZE% = len(CMRQ_TXT$)
830     CMRQ_DATA$ = CMRQ_TXT$(1:CMRQ_DATA_SIZE%)
840   ]

850 Rem *****
860 Rem * Construct a string containing the PLC reference address of *
870 Rem * the COMMREQ status register in STAT_PTR$. *
880 Rem *****
890   STAT_PTR_TYPE% = asc(CMRQ_HDR$(T%))
900   STAT_PTR_OFFSET% = asc(CMRQ_HDR$(O%)) + 256*asc(CMRQ_HDR$(O%+1))
910   Case begin on STAT_PTR_TYPE%
920     Case 8; STAT_PTR$ = "%R"
930     Case 10; STAT_PTR$ = "%AI"
940     Case 12; STAT_PTR$ = "%AQ"
950     Case 16; STAT_PTR$ = "%I"
960     Case 18; STAT_PTR$ = "%Q"
970     Case; STAT_PTR$ = "" ;Rem The status pointer type is
980     Case end ;Rem unrecognized; ignore it.

990 Rem *****
1000 Rem * If a valid status pointer type was received, add the offset *
1010 Rem * to STAT_PTR$. Remember that the offset value in the COMMREQ *
1020 Rem * is zero-based; add 1 to it. The trim$ function removes the *
1030 Rem * leading space which str$ adds when it converts numbers to *
1040 Rem * strings. *
1050 Rem *****
1060   If STAT_PTR$ <> "" then [
1070     STAT_PTR$ = STAT_PTR$ + trim$(str$(STAT_PTR_OFFSET% + 1))
1080     Print "The COMMREQ status pointer is ", STAT_PTR$
1090   ]
1100   Next
1110 Proc end

```

Identifying the Source of Backplane Messages

The **PROCESS_MESSAGE** statement processes backplane messages from three sources: COMMREQ function blocks in the PLC program, messages from the PLC CPU in response to generic requests, and messages sent by another PCM or Series 90 smart module. If your MegaBasic program can receive more than one of these message types, it will need to examine each message to identify it.

All backplane messages contain two fields which can be used to identify them: message type and source. Here is a MegaBasic program fragment showing how to extract message type and source information from a message. Note that the MegaBasic bitwise AND (&) and right shift (>>) operators are available in PCM firmware versions 2.50 and later. Megabasic displays the firmware release as PCM VTOS vX.XX, whenever it starts, where X.XX is the firmware release number. For more information on the message type and source fields, see Table 5-10.

```

100 Dim MESSAGE_HDR$(32)
110 Dim MESSAGE_TXT$(256)

. . .

500 PROCESS_MESSAGE MESSAGE_HDR$, MESSAGE_TXT$
510 MSG_TYPE% = asc(MESSAGE_HDR$(8))
520 SOURCE% = asc(MESSAGE_HDR$(9)) + 256*asc(MESSAGE_HDR$(10))
530 RACK% = SOURCE% & 1111b
540 SLOT% = (SOURCE% >> 4) & 11111b
550 ID% = (SOURCE% >> 9) & 1111111b

```

Line 510 puts the message type value into **MESSAGE_TYPE%**, and line 520 puts the two byte (16 bit) source value into **SOURCE%**. In line 530, the & operator is used to hide or *mask* all but the four least significant bits of **SOURCE%** when they are copied to **RACK%**.

The >> operator is used in line 540 to shift the 5 bits in **SOURCE%** which contain the slot value into the five least significant bits of the result. Then, the & operator assures that only these five bits are copied to **SLOT%**. Similarly, these same operators are used in line 930 to copy just the seven ID bits to **ID%**.

The following table explains the interpretation of **MSG_TYPE%**, **RACK%**, **SLOT%**, and **ID%** values.

Table 5-12. Message Type, Rack, Slot, and ID Values

| Value of MSG_TYPE% | | Data in MESSAGE_TXT\$? | Source of Message |
|----------------------|--------------------|---------------------------|---|
| Decimal | Hexadecimal | | |
| 130 | 82 | Yes | The PLC CPU sent a COMMREQ message from RACK% = 0, SLOT% = 1, and ID% = 9. For more information on COMMREQ messages, see "Interpreting COMMREQ Messages" in this section. |
| 131 | 83 | Yes | |
| 194 | C2 | No | |
| 195 | C3 | No | A generic request message completed successfully. This completion acknowledgment message from the PLC CPU came from RACK% = 0, SLOT% = 1, and ID% = 7. |
| 148 | 94 | Yes | |
| 212 | D4 | No | |
| 209 | D1 | No | A generic request message failed. This PLC message came from RACK% = 0, SLOT% = 1, and ID% = 7. |
| 128-191 Inclusive | 80-BF Inclusive | Yes | If MSG_TYPE% is not one of the specific values listed above, a backplane message was received from another PCM or Series 90 smart module. |
| 192-255 Inclusive | C0-FF Inclusive | No | The RACK% and SLOT% values contain the physical location of the module which sent the message, and the ID% value identifies a particular process within the module. For information on sending messages to another PCM, see "Sending Backplane Messages to Another PCM" in this section. |

The next MegaBasic example program shows how to distinguish between COMMREQ messages and the response messages returned by the PLC CPU when the PCM sends a generic request message. It is used with the **REG_CRQ** ladder program shown earlier in this section.

In line 90, the **GENERIC.PGM** package is accessed from the PCM RAM Disk. **GENERIC.PGM** must be saved to RAM Disk before the example program can be run. The backplane message queue is emptied in lines 160-190, and **USER_BKP_MSG_PROC** is installed as the **BKP_MSG** interrupt procedure in lines 230-250, just as in the previous example.

At line 300, however, the **CHG_PRIV** procedure from **GENERIC.PGM** is used to send a generic message to the PLC CPU. **CHG_PRIV** is described in chapter 5, section 3, *Accessing %P, %L, and Password-Protected Data*.

To make this example shorter and clearer, the COMMREQ status register is assumed to be %R0200. However, the technique of the previous example is better. Here, the **NEXT_CMREQ%** signal is initialized and SYSLINKed in lines 350-360, and the program waits for messages in the loop starting at line 370.

COMMREQ data is printed in lines 440-490, in the same way as in the previous example. If a message is not a COMMREQ, the message type is printed in line 580.

This version of the `USER_BKP_MSG_PROC` procedure, starting at line 630, has an additional complication. It identifies messages by testing the message type field at character position 8. The hexadecimal values 94, D1, and D4 (148, 209, and 212 decimal) identify generic message responses, while 82, 83, C2, and C3 (130, 131, 194, and 195 decimal) identify COMMREQs. To reduce the number of conditions that need to be evaluated by If statements, the MegaBasic bitwise AND operator, `&`, masks the bit that differs between 94 and D4 hexadecimal as well as the two bits that differ among 82, 82, C2, and C3.

Line 820 announces that a generic response arrived, while line 980 announces each COMMREQ. COMMREQ data is processed as in the previous example.

```

10 Rem *****
20 Rem * This example shows how to distinguish between PLC responses *
30 Rem * to generic request messages and COMMREQ messages.          *
40 Rem * The program uses the MegaBasic bitwise AND operator, "&", to *
50 Rem * hide specific bits in an integer value. This feature is    *
60 Rem * available in PCM firmware version 2.50 and later. Note that *
70 Rem * variables used as "&" operands MUST be defined as integers. *
80 Rem *****
90 Access "ram:generic.pgm"
100 Dim RCV_HDR$(32)
110 Dim RCV_TXT$(256)
120 Dim CMRQ_DATA$(256)

130 Rem *****
140 Rem * Throw away messages already in the backplane queue.      *
150 Rem *****
160 Repeat
170   PROCESS_MESSAGE RCV_HDR$, RCV_TXT$
180   If RCV_HDR$ = "" then Exit
190 Next

200 Rem *****
210 Rem * Assign USER_BKP_MSG_PROC to the BKP_MSG interrupt.        *
220 Rem *****
230 Interrupt BKP_MSG, end
240 Interrupt BKP_MSG, USER_BKP_MSG_PROC
250 Interrupt BKP_MSG, on

260 Rem *****
270 Rem * Send a CHG_PRIV generic message to trigger a response    *
280 Rem * message from the PLC CPU.                                *
290 Rem *****
300 CHG_PRIV 3

310 Rem *****
320 Rem * Initialize the NEXT_CMRQ% as the PLC signal. Then, wait for *
330 Rem * two COMMREQs.                                           *
340 Rem *****
350 NEXT_CMRQ% = 1
360 SYSLINK NEXT_CMRQ%, "%R200", UINT
370 Repeat
380   If MSG_TYPE% <> 0 then [

```

```

390 Rem *****
400 Rem * A backplane message arrived. If it is a COMMREQ, print the *
410 Rem * data size and then the data in hexadecimal format. *
420 Rem *****
430   If CMRQ_RCVD% <> 0 then [
440       Print "COMMREQ data size =", CMRQ_DATA_SIZE%, " bytes"
450       Print "COMMREQ data:"
460       For I% = 1 to CMRQ_DATA_SIZE%
470           Print "%3H2", asc(CMRQ_DATA$(I%)),
480       Next I%
490       Print; Print

500 Rem *****
510 Rem * Signal the PLC CPU to send the next COMMREQ. *
520 Rem *****
530     SYSWRITE NEXT_CMRQ%
540   ] Else [

550 Rem *****
560 Rem * A generic message response arrived; print the message type. *
570 Rem *****
580     Print "message type =", MSG_TYPE%; Print
590   ]
600     MSG_TYPE% = 0
610   ]
620 Next

630 Rem *****
640 Rem * This procedure handles the BKP_MSG interrupt. *
650 Rem * Process messages until no more are available. *
660 Rem *****
670 Def proc USER_BKP_MSG_PROC
680   Repeat
690     PROCESS_MESSAGE RCV_HDR$, RCV_TXT$
700     If RCV_HDR$ = "" then Return

710 Rem *****
720 Rem * There is a message. Set CMRQ_DATA_SIZE% to zero. If the *
730 Rem * message is a COMMREQ, change it later. Put the message type *
740 Rem * into MSG_TYPE%. *
750 Rem *****
760     CMRQ_DATA_SIZE% = 0
770     MSG_TYPE% = asc(RCV_HDR$(8))

780 Rem *****
790 Rem * Is the message is a generic message response? *
800 Rem *****
810     If ((MSG_TYPE% & 10111111b) = 94h) or (MSG_TYPE% = 0D1h) then [
820         Print "a PLC generic response message arrived:"

830 Rem *****
840 Rem * Is the message a COMMREQ? Note that the CPU uses the *
850 Rem * MSG_TYPE% values 82, 83, 0C2, and 0C3 hexadecimal only for *
860 Rem * COMMREQs. All four values are detected with one "If" *
870 Rem * statement by ignoring the bits which vary. *
880 Rem *****
890     ] Else If (MSG_TYPE% & 10111110b) = 82h then [

```

```

900 Rem *****
910 Rem * The message is a COMMREQ. It is assumed to be a NOWAIT *
920 Rem * mode COMMREQ. Put the data size into CMRQ_DATA_SIZE%. If *
930 Rem * there are 12 data bytes or less, there is no way to know the *
940 Rem * exact data size unless it is included in the COMMREQ data *
950 Rem * itself. *
960 Rem * Next, copy the COMMREQ data to CMRQ_DATA$. *
970 Rem *****
980 Print "a COMMREQ message arrived:"
990 CMRQ_RCVD% = 1
1000 If RCV_TXT$ = "" then [
1010 CMRQ_DATA_SIZE% = 12
1020 CMRQ_DATA$ = RCV_HDR$(21:12)
1030 ] Else [
1040 CMRQ_DATA_SIZE% = len(RCV_TXT$)
1050 CMRQ_DATA$ = RCV_TXT$(1:CMRQ_DATA_SIZE%)
1060 ]
1070 ]
1080 Next
1090 Proc end

```

Backplane Messages to Another PCM

A MegaBasic program can send backplane messages to the MegaBasic program in another PCM. There are two steps: the sending program simply constructs a message in a string variable. Then, the message is sent by calling the **SEND_MESSAGE** procedure from the MegaBasic PCM extensions. The final example in this section shows how to do it. No PLC ladder program is used.

This example clears the backplane message queue and installs the **USER_BKP_MSG_PROC** procedure, just as in the previous examples. Then it waits five seconds at line 340. Waiting assures that when two PCMs run this same program at about the same time, (with suitable changes to the message destinations), neither one will send a message before the other is ready. If you have two PCMs, try it.

In lines 400-440, a backplane message is initialized and then sent by calling procedure **SEND_UNSOLICITED_MSG**. The destination rack, slot and ID passed to **SEND_UNSOLICITED_MSG** are the address of MegaBasic in the PCM where this program will run. The message type, E0 hexadecimal (224 decimal), was chosen to avoid conflict with PLC CPU messages and to be consistent with the empty **SEND_TXT\$**. Table 5-11 shows the range of valid message type values when the message has extra data in **SEND_TXT\$**, the range which is valid when there is no extra data, and the values which are used by the PLC CPU.

In lines 490-520, another message is sent to the same destination. This message includes 20 characters of text data in **SEND_TXT\$**, so A0 hexadecimal (160 decimal) was chosen as the message type.

After sending the two messages, the program waits for messages in the endless loop at lines 560-570.

The **USER_BKP_MSG_PROC** interrupt procedure in this example, starting at line 580, uses the technique of the previous example to identify each message by testing the type field. If the test at line 750 passes, the message came from the PLC.

Lines 830 and 840 check for messages being returned because they are undeliverable. If you try to send a message to a physically invalid rack/slot combination, it will be returned by the PCM operating system with non-zero error codes in the first two bytes. Messages addressed to a valid rack/slot which is empty, or to a non-existent ID in a rack/slot where there is a PCM, are returned over the backplane. These messages contain 20 hexadecimal (32 decimal) in byte 6.

Finally, messages which are not returned messages or messages from the CPU are processed in lines 900-950. The rack, slot, and ID are extracted from the source field of the message, as described above, and printed.

After identifying the source of the message, `USER_BKP_MSG_PROC` prints it, along with its data string, if any, by calling the `PRINT_MSG` procedure, which is described below.

Procedure `SEND_UNSOLICITED_MSG`, beginning at line 1060, sends the message which was passed as string argument `MSG$`, to the rack, slot and ID destination specified in the `R%`, `I%`, and `ID%` arguments. The bitwise AND (`&`) and left shift (`<<`) operators are used in line 1130 to construct the destination field for the message. The two byte destination value is put into the message, one byte at a time, in lines 1140-1150. Then the message is sent by calling `SEND_MESSAGE`. If the `TXT$` argument is empty, only `MSG$` is passed to `SEND_MESSAGE`; otherwise, both `MSG$` and `TXT$` are passed.

The `PRINT_MSG` procedure prints either a backplane message or a message data string passed in the string argument. It prints the hexadecimal values of the message bytes and, for printable ASCII codes, the character as well. There is quite a bit of extra complexity, but it makes the text data in the main program's messages easy to see.

Lines 1370-1380 complain when an empty string is passed and skip the rest. Line 1400 initializes the line length and the position in `MSG$` of the first character on the first printed line.

The While loop starting at line 1410 does all the work. Line 1420 handles the short printed line that occurs at the end when the length of `MSG$` is not an exact multiple of the line length. Line 1430 prints the hexadecimal values for each line. Line 1440 pads the short line, if any, with enough spaces to make it as long as the preceding lines. Line 1450 prints some spaces and the single quote which introduces the ASCII characters. The For Loop in lines 1460-1530 prints either a character, if it is printable, or a dot if not. Line 1540 prints a trailing single quote and ends the printed line. Line 1550 adjusts the count of characters left to print and the position in `MSG$` of the first character on the next line. Then the While Loop exits if all the characters have been printed; otherwise it continues.

The main program uses separate message header and text strings for messages to be sent on the PLC backplane and messages received from the PLC backplane. They are dimensioned in lines 70-100. Separate strings are necessary to prevent scrambled messages that would occur if one backplane message arrived while the main program was getting ready to send another one.

Programs which send a lot of messages sometimes need to identify them easily. The `SEND_UNSOLICITED_MSG` procedure can add a serial number to the messages it sends. This MegaBasic fragment shows how:

```
1152 SERIAL_NUM% = SERIAL_NUM% + 1
1154 If SERIAL_NUM% >= 256 then SERIAL_NUM% = 1
1156 MSG$(7) = chr$(SERIAL_NUM%)
```

Finally, here is the example program which sends backplane messages between PCMs.

```

10 Rem *****
20 Rem * This example sends two backplane messages to itself. One *
30 Rem * has less than 16 bytes of data; the other has more than 16. *
40 Rem * Try modifying the program to send messages to other modules *
50 Rem * and non-existent modules or IDs. *
60 Rem * *
70 Rem * This program uses the MegaBasic bitwise AND operator, "&", *
80 Rem * the binary shift left operator, "<<", and the binary shift *
90 Rem * right operator, ">>". Note that variables used as operands *
100 Rem * for these operators MUST be defined as integers. They are *
110 Rem * available in PCM firmware version 2.50 and later. *
120 Rem *****
130 Dim RCV_HDR$(32)
140 Dim RCV_TXT$(256)
150 Dim SEND_HDR$(32)
160 Dim SEND_TXT$(256)

170 Rem *****
180 Rem * Throw away messages already in the backplane queue. *
190 Rem *****
200 Repeat
210   PROCESS_MESSAGE RCV_HDR$, RCV_TXT$
220   If RCV_HDR$ = "" then Exit
230 Next

240 Rem *****
250 Rem * Assign USER_BKP_MSG_PROC to the BKP_MSG interrupt. *
260 Rem *****
270 Interrupt BKP_MSG, end
280 Interrupt BKP_MSG, USER_BKP_MSG_PROC
290 Interrupt BKP_MSG, on

300 Rem *****
310 Rem * Wait 5 seconds to give the other PCM (if one is used) time *
320 Rem * to get ready to receive backplane messages. *
330 Rem *****
340 Wait 5

350 Rem *****
360 Rem * Build a backplane message in MSG_HDR$ and send it to ID 3 in *
370 Rem * the module at rack 0, slot 2. Since that is the backplane *
380 Rem * address of this program, an unsolicited message will arrive. *
390 Rem *****
400 SEND_HDR$ = chr$(0) * 32 ;Rem initialize message
410 SEND_HDR$(8) = chr$(0E0h) ;Rem message type
420 SEND_HDR$(17:5) = "HELLO" ;Rem message data
430 SEND_TXT$ = ""
440 SEND_UN SOLICITED_MSG SEND_HDR$, SEND_TXT$, 0, 2, 3

450 Rem *****
460 Rem * Build a backplane message with more than 16 bytes of data. *
470 Rem * Send it to task 3 in this module. *
480 Rem *****
490 SEND_HDR$ = chr$(0) * 32 ;Rem initialize message
500 SEND_HDR$(8) = chr$(0A0h) ;Rem message type
510 SEND_TXT$ = "abcdefghijklmnopqrst"
520 SEND_UN SOLICITED_MSG SEND_HDR$, SEND_TXT$, 0, 2, 3

```

```

530 Rem *****
540 Rem * Repeat forever while messages arrive. *
550 Rem *****
560 Repeat
570 Next

580 Rem *****
590 Rem * This procedure handles the BKP_MSG interrupt. *
600 Rem * Process messages until no more are available. *
610 Rem *****
620 Def proc USER_BKP_MSG_PROC
630   Local MT% ;Rem Message type
640   Local ER% ;Rem Message error
650   Repeat
660     PROCESS_MESSAGE RCV_HDR$, RCV_TXT$
670     If RCV_HDR$ = "" then Return
680     MSG_COUNT% = MSG_COUNT% + 1
690 Rem *****
700 Rem * There is a message. Put the message type into MSG_TYPE%. *
710 Rem * Is the message from the PLC CPU? Check the message type for *
720 Rem * Hexadecimal values 82, 83, 94, 0C2, 0C3, 0D1, and 0D4. *
730 Rem *****
740   MT% = asc(RCV_HDR$(8))
750   If (MT% & 10111111b) = 94h or MT% = 0D1h or (MT% & 0BEh) = 82h then [
760     Print "a message arrived from the PLC CPU:"
770   ] Else [

780 Rem *****
790 Rem * Is it a "return to sender" message? Either bit 5 set to 1 *
800 Rem * in RCV_HDR$(6) or an error code in RCV_HDR$(1) and *
810 Rem * RCV_HDR$(2) indicate a rejected message. *
820 Rem *****
830     ER% = asc(RCV_HDR$(1)) + 256*asc(RCV_HDR$(2))
840     If asc(RCV_HDR$(6)) = 20h or ER% <> 0 then [
850       Print "an outbound message was rejected:"
860     ] Else [

870 Rem *****
880 Rem * Extract the rack, slot, and ID of the sender. *
890 Rem *****
900     SOURCE% = asc(RCV_HDR$(9)) + 256*asc(RCV_HDR$(10))
910     RACK% = SOURCE% & 1111b
920     SLOT% = (SOURCE% >> 4) & 11111b
930     ID% = (SOURCE% >> 9) & 1111111b
940     Print "an unsolicited message arrived from rack", RACK%,
950     Print " slot", SLOT%, " ID", ID%, ":"
960   ]
970 ]

980 Rem *****
990 Rem * Print the message type. Call PRINT_MSG to print the message.*
1000 Rem *****
1010   Print "message type =", MT%
1020   PRINT_MSG RCV_HDR$
1030   If RCV_TXT$ <> "" then PRINT_MSG RCV_TXT$
1040   Next
1050 Proc end

```

```

1060 Rem *****
1070 Rem * This procedure sends the unsolicited backplane message in *
1080 Rem * the MSG$ argument to the Series 90 smart module in the rack, *
1090 Rem * slot, and task ID passed in the R%, S%, and ID% arguments. *
1100 Rem *****
1110 Def proc SEND_UNSOLICITED_MSG MSG$, TXT$, R%, S%, ID%
1120     Local DEST% ;Rem A rack, slot and task ID
1130     DEST% = (R% & 1111b) + ((S% & 11111b) << 4) + ((ID% & 1111111b) << 9)
1140     MSG$(13) = chr$(DEST% mod 256)
1150     MSG$(14) = chr$(DEST% / 256)
1160     If TXT$ = "" then [
1170         SEND_MESSAGE MSG$
1180     ] Else [
1190         SEND_MESSAGE MSG$, TXT$
1200     ]
1210     Return
1220 Proc end

1230 Rem *****
1240 Rem * This procedure prints strings containing binary data as well *
1250 Rem * as text. Each byte is displayed in both hexadecimal and *
1260 Rem * character format; non-printable characters are displayed as *
1270 Rem * ".". Note that 2 byte integer values are printed with the *
1280 Rem * least significant byte on the left. *
1290 Rem *****
1300 Def proc PRINT_MSG MSG$
1310     Local I% ;Rem Position in current line
1320     Local J% ;Rem Starting char position in MSG$ of current line
1330     Local M_LEN% ;Rem Chars from MSG$ left to print
1340     Local L_LEN% ;Rem Length of current line
1350     Local CODE% ;Rem Current ASCII code in MSG$
1360     M_LEN% = len(MSG$)
1370     If M_LEN% = 0 then [
1380         Print "PRINT_MSG error: zero length string"
1390     ] Else [
1400         L_LEN% = 16; J% = 0
1410         While M_LEN% > 0
1420             If L_LEN% > M_LEN% then L_LEN% = M_LEN%
1430             For I% = 1 to L_LEN%; Print "%3H2", asc(MSG$(I%+J%)); Next I%
1440             For I% = L_LEN% + 1 to 16; Print " "; Next I%
1450             Print " ",
1460             For I% = 1 to L_LEN%
1470                 CODE% = asc(MSG$(I% + J%))
1480                 If (CODE% < asc(" ")) or (CODE% > asc("~")) then [
1490                     Print ".",
1500                 ] Else [
1510                     Print chr$(CODE%),
1520                 ]
1530             Next I%
1540             Print ""
1550             M_LEN% -= L_LEN%; J% += L_LEN%
1560         Next
1570         Print
1580     ]
1590     Return
1600 Proc end

```


Section 10: Asynchronous Serial Input and Output

Most of the MegaBasic statements and functions for reading and writing devices cause the MegaBasic program to wait until the input or output has completed. During this time, all MegaBasic execution stops, including backplane interrupt servicing and Control-C processing. In order to provide fast response time to interrupts, it is a good idea to keep inputs and outputs small in length. At 19.2K bits per second, it takes 0.5 milliseconds to send a single character.

For applications where standard I/O causes timing problems, the program can elect to do **NOWAIT I/O**. There are five **NOWAIT I/O** statements:

- **NOWAIT_OPEN**
- **NOWAIT_CLOSE**
- **NOWAIT_READ**
- **NOWAIT_WRITE**
- **NOWAIT_SEEK**

Beginning with Release 2.51, there are two additional **NOWAIT I/O** statements:

- **NOWAIT_READ_ABORT**
- **NOWAIT_WRITE_ABORT**

Unlike normal MegaBasic I/O statements, the **NOWAIT** statements, except for **NOWAIT_OPEN**, do not wait for their operation to complete before allowing the MegaBasic program to continue. In addition, **NOWAIT_READ** and **NOWAIT_WRITE** can cause the MegaBasic program to be interrupted when the operation completes. This allows the MegaBasic program to do remote file access, collect input, and do output without sacrificing program speed or responsiveness.

Note

The one limitation to **NOWAIT I/O** processing is that only strings can be read and written. This is because the **NOWAIT I/O** statements bypass the normal I/O formatting that is done by the MegaBasic interpreter's **PRINT** statement. The strings can be formatted using other MegaBasic functions such as **STR\$**.

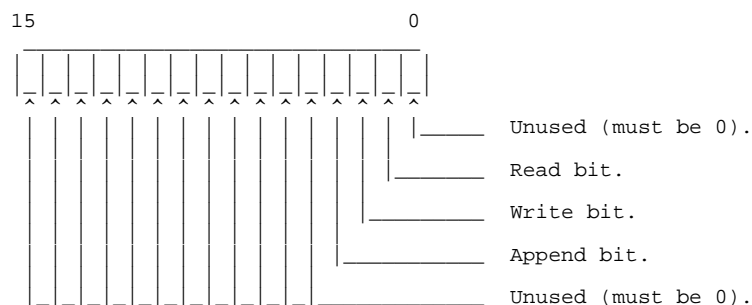
NOWAIT_READ and **NOWAIT_WRITE** are unnecessary for the RAM Disk and the NULL device because these accesses always complete immediately. **NOWAIT_READ** and **NOWAIT_WRITE**, therefore, are not recommended for these devices.

NOWAIT_OPEN

The **NOWAIT_OPEN** function opens a channel for subsequent asynchronous communication. Its format is:

```
handle = NOWAIT_OPEN ( <"name">, <mode>, [buffer size], [error] )
```

| Argument | Description |
|----------|---|
| Name | The name of the device to be read or written. It may be a device, such as COM1 : , or a file, such as PC:data.out . If it is a file name, the name must be fully qualified as device:filename . For a device, the name can be a literal string or a string variable containing a properly formatted name. |
| Mode | Specifies how the file is to be opened. It is formatted as shown below. <div> <p><u>Read Bit:</u> If set to 1, the channel for the device or file has read privilege. If the file does not exist, an error is returned.</p> <p><u>Write Bit:</u> If set to 1, the channel for the device or file has read and write privilege. If a file already exists, its contents are erased. If it does not exist, the file is created.</p> <p><u>Append Bit:</u> If set to 1, the channel for the device or file has read and write privilege. If the file already exists, its contents are not erased. If it does not exist, an error is returned.</p> </div> |



| Argument | Description |
|-------------|---|
| Buffer Size | <p>When a NOWAIT_READ or NOWAIT_WRITE is done, an intermediate buffer is used to transfer the data between the MegaBasic program and the device. The buffer size argument specifies the maximum total number of bytes that can be buffered at once. This may span several outstanding NOWAIT_READ or NOWAIT_WRITE statements. If this value is omitted or set to zero, the amount of data that can be buffered is limited only by the amount of free memory in the PCM.</p> <p>Specifying a maximum buffer size does not actually cause a buffer to be allocated. It only places a limit on the total size of subsequent outstanding allocations. That is, NOWAIT_READs and NOWAIT_WRITEs are not being satisfied as fast as new buffers (additional requests) are allocated. It is possible to run out of memory on the PCM before the maximum buffer size is reached. OPEN does, however, verify that the maximum buffer size is available at the time that the NOWAIT_OPEN call is made.</p> |
| Error | <p>Determines what action to take if subsequent transfers exceed the maximum buffer size or cause the PCM to run out of memory. If the error flag is omitted or set to zero, the MegaBasic program waits until memory becomes available and then continues. Otherwise, a memory error is generated.</p> |

NOWAIT_OPEN returns a handle for the channel that is an integer in the range 0 through 65,535. This handle must be saved in order to use it with the other **NOWAIT** statements.

NOWAIT_READ and NOWAIT_WRITE

The **NOWAIT_READ** and **NOWAIT_WRITE** statements handle the movement of data between the MegaBasic program and various devices. Their formats are:

```
NOWAIT_READ <channel handle>, <str_vbl$>, [xfer handle]
NOWAIT_WRITE <channel handle>, <str_vbl$>, [xfer handle]
```

| Argument | Description |
|-----------------|--|
| ChannelHandle | An integer that was returned as the result of a previous NOWAIT_OPEN call. |
| String Variable | A MegaBasic string variable that has already been defined in the program. The length of this string variable may up to 65,535 bytes. The number of bytes read or written depends upon the current size of the string variable. |
| TransferHandle | An optional integer number in the range 0 through 65,535 that identifies a particular transfer. When the transfer completes, an interrupt is generated. This number is then made available to the interrupt procedure. |

When a **NOWAIT_READ** or **NOWAIT_WRITE** completes, an interrupt is posted to the interpreter. If there is no interrupt procedure defined, the interrupt is ignored. In most cases, however, it is advantageous to use interrupt procedures.

If user procedures are supplied for the **NOWAIT_READ** or **NOWAIT_WRITE** interrupts, the old interrupt definition must be erased and a new definition activated by the user program before the new procedures are used, much the same as for backplane interrupt procedures. For example:

```
100 Interrupt NOWAIT_RD, end
110 Interrupt NOWAIT_RD, USER_ISR_PROC
120 Interrupt NOWAIT_RD, on
```

where **USER_ISR_PROC** is the name of the user procedure that handles **NOWAIT_READ** processing.

Multiple **NOWAIT_READS** and **NOWAIT_WRITES** can be done simultaneously over a single channel, as long as the maximum buffer space for the channel is not exceeded. The transfers are processed in the order that the statements are executed.

If an interrupt is defined for **NOWAIT_RD** or **NOWAIT_WR** (5 and 4) and the interrupt occurs, the MegaBasic interrupt procedure must use either the **PROCESS_READ** or **PROCESS_WRITE** statement to complete the processing and obtain a pointer to the variable, the handle and an error code. The formats of the **PROCESS_READ** and **PROCESS_WRITE** statements are:

```
PROCESS_READ <vbl_ptr>, <handle>, <error>
PROCESS_WRITE <vbl_ptr>, <handle>, <error>
```

| Argument | Description |
|-----------------|---|
| VariablePointer | A MegaBasic pointer. After the statement is executed, the pointer points to the string variable that was transferred. |
| Handle | Contains the transfer handle passed by the NOWAIT_READ or NOWAIT_WRITE statement that initiated the transfer. If none was passed, the handle argument is set to zero. |
| Error | Set to an error code or zero when there is no error. |

This example is a **NOWAIT_READ** interrupt procedure using **PROCESS_READ**:

```

500 Def Proc USER_NOWAIT_RD
510 Local VBLPTR%, HANDLE%, ERR
520 Repeat
530   PROCESS_READ VBLPTR%, HANDLE%, ERR
540   If VBLPTR% = 0 then Return
550   If ERR then PROCESS_ERR ERR
560   Rem Process data
570 Next
580 Proc end

```

Beginning with PCM firmware version 3.00, **NOWAIT_READ** operations on COM1 and COM2 may be terminated by receipt of a specific character. This is done using a slightly different **NOWAIT_OPEN** for the COM port. For example:

```

I%=NOWAIT_OPEN("COM1:13",6)
DIM A$(20)
NOWAIT_READ I%,A$

```

This **NOWAIT_READ** will complete if a <CR> (CHR\$(13)) is received or when twenty characters are received. The termination character is placed into the string before the string is returned to the MegaBasic program. The current length of the string will reflect the actual number of characters received.

Serial Port Error Codes Returned by PROCESS_READ

When a serial port error occurs in a **PROCESS_READ** statement, the variable specified by the Error argument will contain a non-zero value. This value will consist of one, two, or three bits OR'ed together:

| Code | Status |
|------|----------------|
| 10H | Parity error. |
| 20H | Overrun error. |
| 40H | Framing error. |

Note

These codes are different than the ones listed in section 1 of this chapter, which apply to the MegaBasic **INCHR\$** function.

The **PROCESS_ERR** procedure called in line 530 of the example above, is not shown. The specific actions required to recover from serial port errors will depend on the application.

NOWAIT_CLOSE

The **NOWAIT_CLOSE** statement is used to close a channel that has been opened for **NOWAIT** access. Once a channel is closed, subsequent **NOWAIT_READS** and **NOWAIT_WRITES** to the channel result in errors, although all outstanding transfers are completed. The format of **NOWAIT_CLOSE** is:

```
NOWAIT_CLOSE <channel handle>
```

NOWAIT_SEEK

The **NOWAIT_SEEK** statement is used for setting the file pointer of a **NOWAIT** channel. This is mainly used when accessing files over the PCM's remote file server. Its format is:

```
NOWAIT_SEEK <channel handle>, <position>
```

Since the completion of a **NOWAIT_SEEK** does not generate an interrupt, any error that occurs on a **NOWAIT_SEEK** is not recognized until a subsequent **NOWAIT_READ** or **NOWAIT_WRITE** is done. Depending on the device, a **NOWAIT_WRITE** beyond the end of file may or may not be considered an error. For a **NOWAIT_READ**, an end of file error is returned in the error variable of a **PROCESS_READ** call. In addition, the string variable contains only those bytes read before the end of the file.

Care must be taken when stopping and restarting programs that do **NOWAIT I/O**, particularly **NOWAIT_READS**. If a program is in the middle of a **NOWAIT_READ** or **NOWAIT_WRITE** when it stops, it tries to finish the operation before it runs again. In the case of a device that is not responding, this can cause MegaBasic to wait indefinitely or until the PCM is reset.

Another consideration for **NOWAIT I/O** programming occurs when a PC is attached to the PCM and is being used as both a file server and a terminal. In this case the PC: device and the COM: port to which it is attached should not be accessed in **NOWAIT** mode unless it is certain that file traffic and terminal traffic do not occur at the same time. Otherwise, terminal traffic will interfere with the file server protocol.

NOWAIT_READ_ABORT

The `NOWAIT_READ_ABORT` statement is used to prematurely stop any `NOWAIT_READS` that are active on a particular channel. If interrupts are being used for the `NOWAIT_RD` handling, an interrupt will be generated to signal the completion of the `NOWAIT_READ`.

The `<error>` field of the `PROCESS_READ` statement will indicate that the transfer was aborted. The current length of the string will indicate how many characters were received before the abort. Typically, an application will program a timer at the beginning of a `NOWAIT_READ`. If the timer elapses before the `NOWAIT_READ` completes, then the application stops the `NOWAIT_READ` with a `NOWAIT_READ_ABORT`. The format of a `NOWAIT_READ_ABORT` is:

```
NOWAIT_READ_ABORT <channel handle>
```

All `NOWAIT_READ` operations active on the channel will be aborted by this statement.

NOWAIT_WRITE_ABORT

The `NOWAIT_WRITE_ABORT` statement is used to prematurely stop any `NOWAIT_WRITES` that are active on a particular channel. Typically, a `NOWAIT_WRITE_ABORT` would be used to flush output upon receipt of a break or upon detection of a loss of connection (no DCD/CTS TRUE status). If interrupts are being used for the `NOWAIT_WR` handling, an interrupt will be generated to signal the completion of the `NOWAIT_WRITE`.

The `<error>` field of the `PROCESS_WRITE` statement will indicate that the write was aborted. The format of a `NOWAIT_WRITE_ABORT` is:

```
NOWAIT_WRITE_ABORT <channel handle>
```

All `NOWAIT_WRITE` operations active on the channel will be aborted by this statement.

Example NOWAIT Program

The following example program illustrates the use of **NOWAIT** statements for serial I/O:

```

10 Rem *** Program to exercise NOWAIT functions of MegaBasic for serial
20 Rem *** input and output.
30 Access "ram:vt100.pgm"
40 Dim TEXT_STR1$(80)
50 Dim TEXT_STR2$(80)
60 Dim TEXT_STR3$(80)

70 Def integer HANDLE1
80 Def integer HANDLE2
90 Def integer HANDLE3
100 Interrupt NOWAIT_RD, end
110 Interrupt NOWAIT_WR, end
120 Interrupt NOWAIT_RD, READ_INT
130 Interrupt NOWAIT_WR, WRITE_INT
140 Interrupt on
150 HANDLE1 = NOWAIT_OPEN ( "COM1:", 0)
160 HANDLE2 = NOWAIT_OPEN ( "COM1:", 0)
170 HANDLE3 = NOWAIT_OPEN ( "COM1:", 0)
180 Len ( TEXT_STR1$ ) = 5
190 Len ( TEXT_STR2$ ) = 5
200 Len ( TEXT_STR3$ ) = 5
210 NOWAIT_READ HANDLE1, TEXT_STR1$, 1
220 NOWAIT_READ HANDLE1, TEXT_STR2$, 2
230 NOWAIT_READ HANDLE1, TEXT_STR3$, 3
240 CLS

250 Rem *** Enter non-terminating loop
260 Repeat
270 Next

280 Def proc READ_INT
290   Local VBLPTR%, HANDLE%, XFERERR%
300   Repeat
310     PROCESS_READ VBLPTR%, HANDLE%, XFERERR%
320     If VBLPTR% = 0 then Return
330     If XFERERR% <> 0 then Stop "Transfer error"
340     NOWAIT_WRITE HANDLE2, *VBLPTR%, HANDLE%
350   Next
360 Proc end

370 Def proc WRITE_INT
380   Local VBLPTR%, HANDLE%, XFERERR%
390   Repeat
400     PROCESS_WRITE VBLPTR%, HANDLE%, XFERERR%
410     If VBLPTR% = 0 then Return
420     If XFERERR% <> 0 then Stop "Transfer error"
430     Print HANDLE%
440     NOWAIT_READ HANDLE1, *VBLPTR%, HANDLE%
450   Next
460 Proc end

```

This example program uses two interrupt procedures, **READ_INT** and **WRITE_INT**, to process **NOWAIT** serial I/O. The main program simply initializes program variables, opens serial port 1, assigns the interrupt procedures to the **NOWAIT_RD** and **NOWAIT_WR** interrupts, makes three **NOWAIT_READ** requests, and then goes into the non-terminating loop at line 260.

When one of the main program's **NOWAIT_READ** requests completes, **READ_INT** is called to process the data. **READ_INT** uses a **NOWAIT_WRITE** statement to send the new data back out port 1.

Try running this program yourself. First, load **VT100.PGM** into MegaBasic from the **\PCOP\EXAMPLES.PCM** directory created when you installed TERMF or PCOP. Save **VT100.PGM** to the PCM RAM disk. Then, load the example program and run it. Each time you type five characters, they will be received by **READ_INT** and echoed back to your PC display by **NOWAIT_WRITE**. When the write operation completes, **WRITE_INT** will be called. It will print the handle value of the **NOWAIT_READ** request which received your five characters.

Section 11: VME Functions

Series 90-70 PLC ladder program can also communicate with the PCM using the VME Read (**VMERD**) and VME Write (**VMEWRT**) functions. These functions are not available for Series 90-30 PLCs.

VMERD, **VMEWRT**, and other associated functions treat the PCM as a standard VME board. Data is moved to and from the PCM's VME bus dual port RAM.

VME functions usually execute faster than the equivalent COMMREQ for the same data transfer. These functions can also be useful in situations when the PLC System Communications Window must be severely shortened or eliminated.

However, many of the advantages of the COMMREQ are not present for a **VMERD** or **VMEWRT**, including:

- Guaranteed data coherency.
- Automatic protection against simultaneous access to the same data location by the PCM and PLC CPU.
- Fault reporting of some user programming errors, such as an invalid task or the wrong rack/slot.
- No requirement for user knowledge or manipulation of dual port addresses.

Most applications should use the COMMREQ function to transfer data between the ladder program and the PCM. The VME functions should be used only when timing constraints or other factors dictate their use.

VME Function Blocks for Communicating with the PCM

A group of PLC functions blocks is available in Logicmaster 90 software to allow a Series 90-70 PLC CPU to communicate with VME modules, including the PCM. These functions include:

- VME Read (**VMERD**).
- VME Write (**VMEWRT**).
- VME Read/Modify/Write (**VMERMW**).
- VME Test and Set (**VMETS**).

Each of these function blocks is discussed in detail later in this section.

Some Rules for VME Bus Operations in Series 90-70 PLCs

VME bus block move transfers are not supported by Series 90-70 PLCs.

Do not place the PCM, or any other GE Fanuc board, in a standard VME rack. GE Fanuc boards must be installed only in Series 90-70 PLC racks. For more information about VME in the Series 90-70 PCM, refer to the *Series 90-70 Programmable Controller User's Guide to the Integration of Third Party VME Modules*, GFK-0448.

General VME Information for the PCM

When the PCM is used for VME functions, it should be configured in the Logicmaster 90 configuration package. That is, configure the PCM in the same way it would be selected for non-VME functions. The PCM should not be configured as a foreign VME module.

Addresses on the Series 90-70 VME bus consist of two parts, an address modifier (AM) code and a 24 bit address. The AM code consists of 6 bits and is used to select the type of VME access (e.g., the number of address bits used). The AM code for the PCM is 39H. It specifies the **StandardNon-privileged** access type.

VME bus addresses for PCM modules used as VME function block targets depend on the rack and slot location of the PCM. The PCM must be addressed in the range allocated to the rack and slot where it is located. Address allocations for PCMs are provided in the following table.

Table 5-13. PCM Address Allocation by Slot and Rack for Standard Non-Privileged Access - 39H

| Rack Number | Slot Number | | | | | | | |
|-------------|---|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 00000H to 07FFFH | 02000H to 027FFFH | 04000H to 047FFFH | 06000H to 067FFFH | 08000H to 087FFFH | 0A000H to 0A7FFFH | 0C000H to 0C7FFFH | 0E000H to 0E7FFFH |
| 0 | 10000H through 7FFFFH; User Defined for Rack 0 Only | | | | | | | |
| 1 | E0000H to E07FFFH | E2000H to E27FFFH | E4000H to E47FFFH | E6000H to E67FFFH | E8000H to E87FFFH | EA000H to EA7FFFH | EC000H to EC7FFFH | EE000H to EE7FFFH |
| 2 | D0000H to D07FFFH | D2000H to D27FFFH | D4000H to D47FFFH | D6000H to D67FFFH | D8000H to D87FFFH | DA000H to DA7FFFH | DC000H to DC7FFFH | DE000H to DE7FFFH |
| 3 | C0000H to C07FFFH | C2000H to C27FFFH | C4000H to C47FFFH | C6000H to C67FFFH | C8000H to C87FFFH | CA000H to CA7FFFH | CC000H to CC7FFFH | CE000H to CE7FFFH |
| 4 | B0000H to B07FFFH | B2000H to B27FFFH | B4000H to B47FFFH | B6000H to B67FFFH | B8000H to B87FFFH | BA000H to BA7FFFH | BC000H to BC7FFFH | BE000H to BE7FFFH |
| 5 | A0000H to A07FFFH | A2000H to A27FFFH | A4000H to A47FFFH | A6000H to A67FFFH | A8000H to A87FFFH | AA000H to AA7FFFH | AC000H to AC7FFFH | AE000H to AE7FFFH |
| 6 | 90000H to 907FFFH | 92000H to 927FFFH | 94000H to 947FFFH | 96000H to 967FFFH | 98000H to 987FFFH | 9A000H to 9A7FFFH | 9C000H to 9C7FFFH | 9E000H to 9E7FFFH |
| 7 | 80000H to 807FFFH | 82000H to 827FFFH | 84000H to 847FFFH | 86000H to 867FFFH | 88000H to 887FFFH | 8A000H to 8A7FFFH | 8C000H to 8C7FFFH | 8E000H to 8E7FFFH |

* Rack 0 is the CPU rack.

PCM Dual Port RAM Available for Applications

The PCM system software uses the first 4000h bytes of the PCM dual port RAM.

Caution

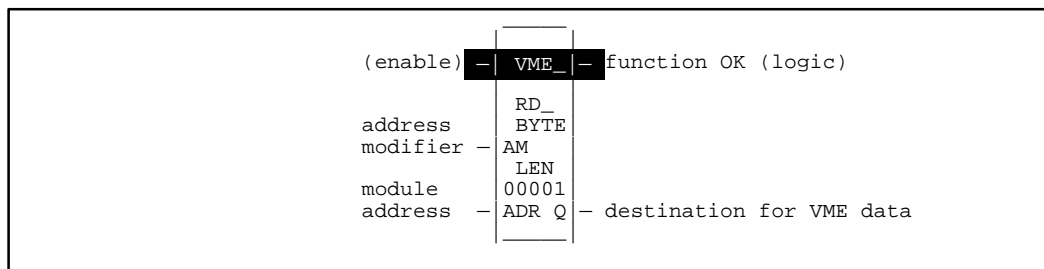
Addresses within the first 4000h should never be used for user VME communication with the PCM. Use only addresses at or above 0A4000h.

Future PCM enhancements are likely to use dual port RAM above the first 4000h bytes. Applications should use the highest possible address for VME communication.

VME Read Function

The **VMERD** function can be used to read data from the dual port RAM of the Series 90-70 PCM to the CPU. This function should be executed before the data is needed in the PLC ladder program.

The format of the **VMERD** function block is:



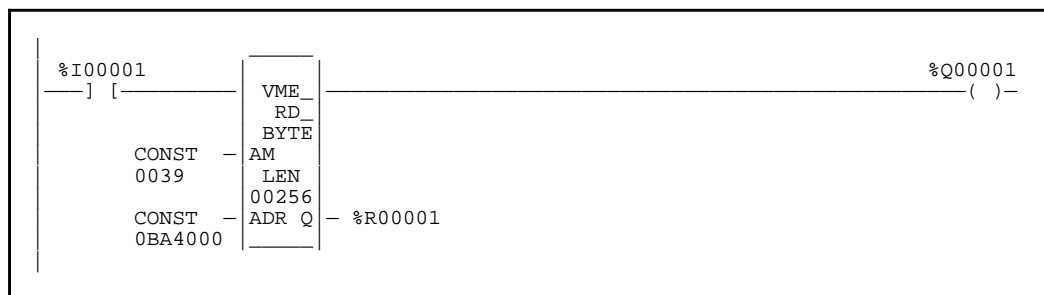
| Parameter | Description |
|-----------------|---|
| Enable | Power flow input that, when true, enables the execution of the function. |
| Type | Function type, either byte or word, to select the corresponding type of VME bus access to be performed. |
| Length | An internal parameter that, depending on the function type, specifies the number of bytes or words to be transferred. |
| AddressModifier | Hexadecimal value coded to specify the address modifier for the PCM. The PCM AM code is always 39H (see above). |
| Address | A double word specifying the hexadecimal address where the first word or byte is read from the VME bus. It may be a constant or the reference address of the first (low) word of two words containing the module address. The address is based on the rack and slot where the PCM is located. (Refer to “Address Allocation by Rack and Slot” in this section.) |
| OK | Power flow output that is true when the function is enabled and completes successfully |
| Q | Specifies the first PLC user reference location into which the data read from the PCM is to be stored. |

When the **VMERD** function receives power flow through its ENABLE input, the function accesses the PCM at the specified address ADR and copies LEN data units (words or bytes) from the PCM to PLC locations beginning at the output parameter Q. When the operation is successfully completed, the **VMERD** function passes power to the right through the OK output.

For information on PCM module addressing using addresses and address modifier codes, refer to “General VME Information for the PCM,” presented earlier in this section.

Example VMERD Function

In the following example, 256 bytes of data are read from a PCM in rack 4, slot 7 into registers %R00001 through %R00128 when enabling input %I00001 goes true. Unless an error occurs while reading the data, output %Q00001 is set to true.



In Series 90-70 PCMs, VME dual port memory occupies 32K bytes, starting at address 0A0000h:0000h, regardless of the rack and slot where the PCM is installed. In this example, the VME bus address 0BA7000h corresponds to the PCM internal address 0A000h:7000h.

There are several ways a MegaBasic program could move data to this address in VME dual port RAM. One of the simplest ways is to use the **FILL** statement:

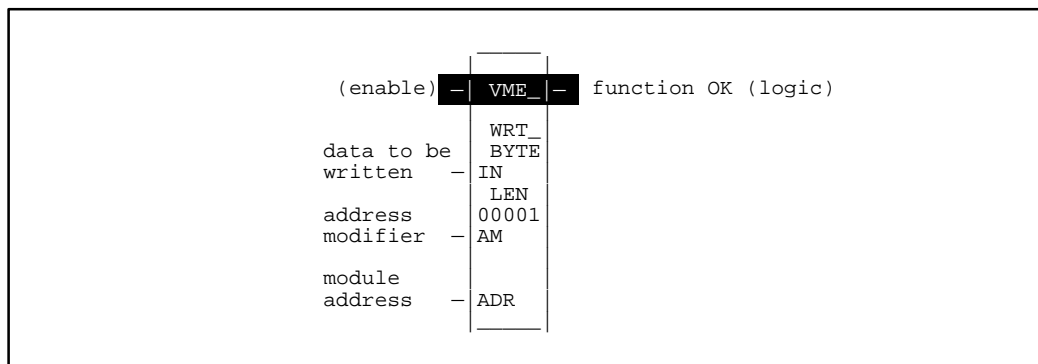
```
FILL 0A000h:7000h, <data list>
```

where **<data list>** is a list of data items expressed as MegaBasic variables, constants, or expressions. For more information, refer to the *MegaBasic Programming Language Reference Manual*, GFK-0256.

VME Write Function

The **VMEWRT** function can be used to write data from the Series 90-70 CPU to the VME dual port RAM of the PCM. Locate the function block at a place in the program where the output data is ready to send.

The format of the **VMEWRT** function block is:



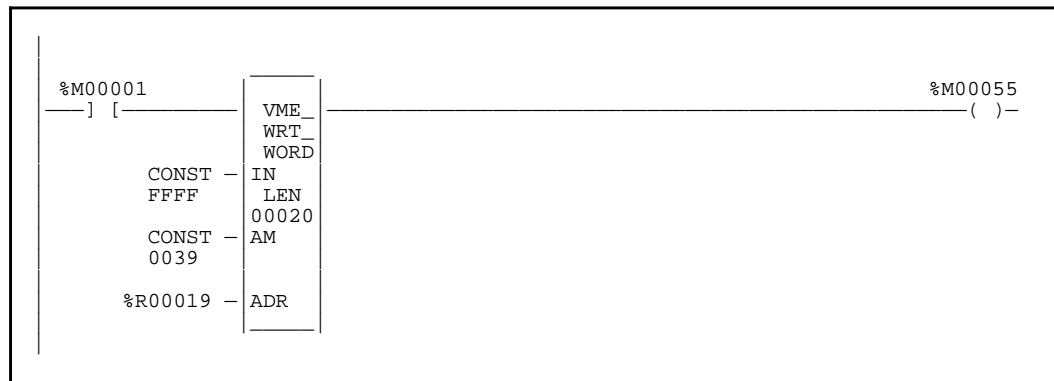
| Parameter | Description |
|-----------------|--|
| Enable | Power flow input that, when true, enables the execution of the function. |
| Type | Function type, either byte or word, to select the corresponding type of VME bus access to be performed. |
| Length | An internal parameter that, depending on the function type, specifies the number of bytes or words to be transferred. |
| In | Specifies the first PLC user reference location where the data to be written to the PCM is stored. This parameter may be a constant, in which case the constant value is written to all VME addresses covered by the function's length. |
| AddressModifier | Hexadecimal value coded to specify the address modifier of the PCM. The PCM AM code is always 39H (see above). |
| Address | A double word specifying the hexadecimal address where the first word or byte is written to the VME bus. It may be a constant or the reference address of the first (low) word of two words containing the module address. The address is based on the rack and slot where the PCM is located. Refer to "Address Allocation by Rack and Slot" in this section. |
| OK | Power flow output that is true when the function is enabled and completes successfully |

When the **VMEWRT** function receives power flow through its enable input, LEN data units (words or bytes) from the PLC locations beginning at the input parameter IN are written to the PCM at the specified address ADR. When the operation is successfully completed, the **VMEWRT** function passes power to the right through the OK output.

For information on PCM module addressing using address and address modifier codes, refer to "General VME Information for the PCM," presented earlier in this section.

Example VMEWRT Function

In the following example, the hexadecimal value FFFF is written to each of 20 words on the PCM during every sweep when enabling input %M00001 is true. The starting (lowest) PCM address is specified by the contents of %R00019 (low word) and %R00020 (high word). Unless an error occurs while writing the data, internal coil %M00055 is set to true.

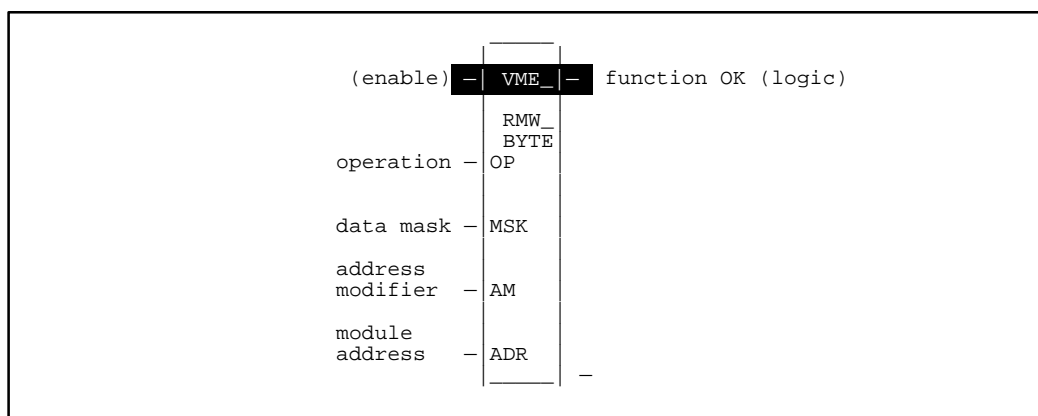


The PCM must be located in the rack and slot corresponding to the address contained in %R00019 and %R00020. The MegaBasic user program would read this data from the PCM dual port RAM, using an **EXAM** statement at the appropriate address.

VME Read/Modify/Write Function

The **VMERMW** function can be used to update a data element in the dual port RAM of the Series 90-70 PCM.

The format of the **VMERMW** function block is:



| Parameter | Description |
|-----------------|--|
| Enable | Power flow input that, when true, enables the execution of the function. |
| Type | Function type, either byte or word, to select the corresponding type of VME bus access to be performed. |
| Operation | A constant which specifies whether an AND or OR function is to be used to combine the data and the mask. 0 specifies AND; 1 specifies OR. |
| Mask | A word value containing a mask to be ANDed or ORed with the data read from the bus. If the type is byte, only the low 8 bits of the mask are used. |
| AddressModifier | Hexadecimal value coded to specify the rack in which the module resides and the access mode of the VME bus access to be performed. |
| Address | A double word specifying the hexadecimal address of the word or byte to be accessed. It may be a constant or the reference address of the first (low) word of two words containing the module address. The address is based on the rack and slot where the module is located. Refer to "AddressAllocation by Rack and Slot" in this section. |
| OK | Power flow output that is true when the function is enabled and completes successfully |

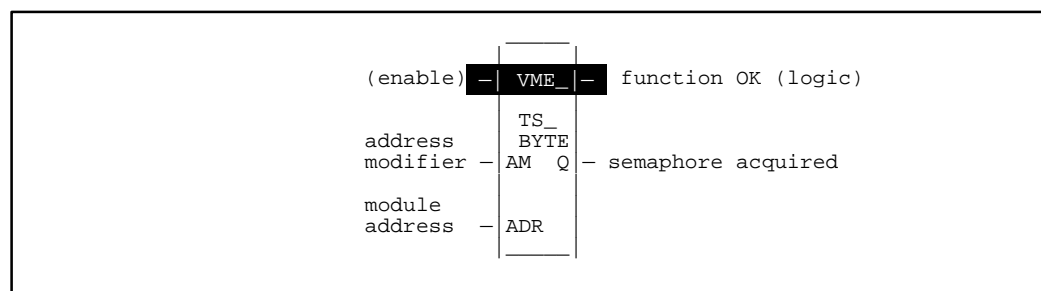
When the **VMERMW** function receives power flow through its enable input, the function reads a word or byte of data from the module at the specified address (ADR) and address modifier (AM). This byte or word of data is combined (AND/OR) with the data mask (MSK). Selection of AND or OR is made using the input OP. If byte data is specified, only the lower 8 bits of MSK are used. The result is then written back to the same VME address from which it was read. When the operation is successfully completed, the **VMERMW** function passes power to the right through the OK output.

For information on PCM module addressing using address and address modifier codes, refer to "General VME Information for the PCM," presented earlier in this section.

VME Test and Set Function

The **VMETS** function can be used to handle semaphores located in the dual-port RAM of the Series 90-70 PCM. The **VMETS** function exchanges a boolean true (1) for the value currently at the semaphore location. If that value already was true, then the **VMETS** function does not acquire the semaphore. If the existing value was false, the semaphore is set and the **VMETS** function block has control of the semaphore and the use of the memory area it controls. The semaphore is cleared and ownership relinquished by using the **VMEWRT** function to write false (0) to the semaphore location.

The format of the **VMETS** function block is:



| Parameter | Description |
|-----------------|---|
| Enable | Power flow input that, when true, enables the execution of the function. |
| Type | Function type, either byte or word, to select the corresponding type of VME bus access to be performed. |
| AddressModifier | Hexadecimal value coded to specify the rack in which the module resides and the access mode of the VME bus access to be performed. |
| Address | A double word specifying the hexadecimal address of the first word or byte to be accessed. It may be a constant or the reference address of the first (low) word of two words containing the module address. The address is based on the rack and slot the module is located. Refer to "Address Allocation by Rack and Slot" in this section. |
| OK | Power flow output that is true when the function is enabled and completes successfully |
| Q | Set to true if the semaphore was acquired. Set to false if the semaphore was not available, i.e., was owned by another task. |

When the **VMETS** function receives power flow, a boolean true is exchanged with the data at the address specified by ADR using the address mode specified by AM. The **VMETS** function sets the Q output to true if the semaphore (false) was available and acquired. When the operation is successfully completed, the **VMETS** function passes power to the right through the OK output.

For information on PCM module addressing using address and address modifier codes, refer to "General VME Information for the PCM," presented earlier in this section.

For more information on Series 90-70 PLC programming, refer to the *Logicmaster 90-70 Programming Software User's Manual*, GFK-0263, and the *Series 90-70 Programmable Controller Reference Manual*, GFK-0265. For more information about Series 90-70 VME bus applications, refer to the *Series 90-70 Programmable Controller User's Guide to the Integration of Third Party VME Modules*, GFK-0448.

MegaBasic Program Access to PCM Dual Port RAM

The PCM memory map places the dual port RAM in the address range 0A0000h to 0A7FFFh for the current revision of the Series 90-70 PCM.

The PCM User MegaBasic program uses **EXAM** and **FILL** statements to access the PCM Dual Port RAM. The **EXAM** statement is used to read data from a specified location in the PCM dual port RAM. The **FILL** statement is used to write to a location in the dual port RAM.

The format of the **EXAM** statement is:

```
EXAM st_addr, var_list
```

| Parameter | Description |
|-----------------|---|
| StartingAddress | The starting address of the data. |
| VariableList | The list of variables, separated by commas. |

The format of the **FILL** statement is:

```
FILL st_addr, data_list
```

| Parameter | Description |
|-----------------|--|
| StartingAddress | The starting address of the data. |
| Data List | The list of data items, separated by commas. |

The address of the data in the PCM dual port RAM is usually represented as a segmented address using two words, segment and offset.

Example **EXAM** and **FILL** statements:

```
EXAM 0A000h:4000h, @DATA
FILL 0A000h:605Ah, @1
```

Note

The @ character in the examples above denotes word (16-bit) access to the memory location.

For more information on the **EXAM** and **FILL** statements, refer to the *MegaBasic Programming Language Reference Manual*, GFK-0256.

The next section of this chapter contains an example of PLC VME functions and MegaBasic **EXAM** and **FILL** statements.

Section 12: Programming Example using VME Functions

The following programming example uses the VME functions of the Series 90-70 PLC to communicate with a PCM. The PCM is located in slot 3 of rack 0, corresponding to the address programmed in the VME functions. Modules in that rack and slot location may use addresses 020000H-027FFFH as shown in table 5-4 in section 4 of this chapter. Note that the AM code is always 39H. A semaphore, located at address 0A4000H in the PCM, is used to prevent simultaneous access to the data by the PCM and the PLC.

The address programmed in the VME functions indicates rack, slot, and data offset in the PCM dual port RAM. The segment word on the PCM side is always 0A0000h. Allowable offsets are in the range 4000h-7FFFh. (Refer to the preceding section in this chapter.) The 4000h in the VME address corresponds to the offset 4000h in PCM dual port RAM.

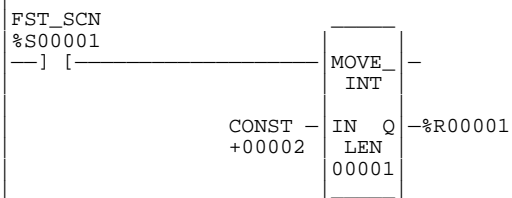
In the example ladder, when %R1 contains 2, the semaphore at offset 4000h on the PCM is read. If the semaphore has the value 1, the value of %R1 is incremented to 3. If the semaphore is 0, %R1 remains 2. As long as %R1 is 2, the VME read of the semaphore occurs on every sweep.

When %R1 contains 3, the program writes 22 bytes to the PCM from %R20 using the VMEWRT function block. The data is placed at offset 4002h in PCM dual port RAM. Note that the semaphore (offset 4000h) is immediately cleared after the VMEWRT. %R1 is changed to 2 to permit the operation to be repeated.

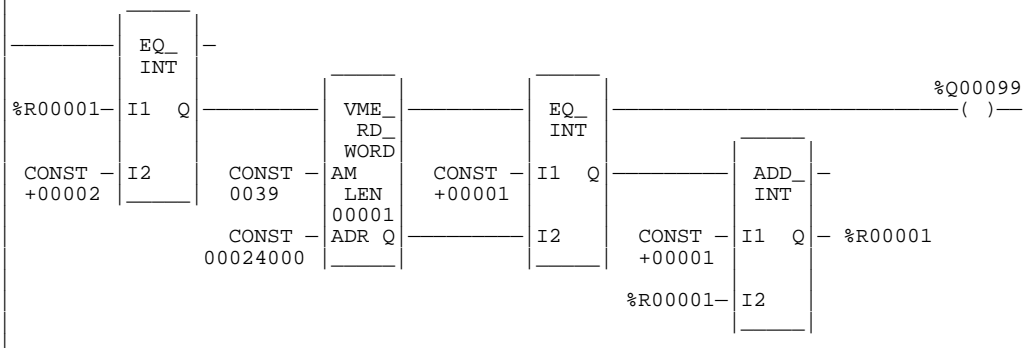
The MegaBasic program sets up variables containing the locations of the data in dual port RAM: DPR_SEG, FLAG_OFFSET and DPR_OFFSET. The program writes 1 to the semaphore location using the **FILL** statement to start processing data. As long as the semaphore value is 1, the MegaBasic program remains in a loop waiting for the PLC to complete its transfer of data. When the semaphore becomes 0, the program uses the **EXAM** statement to move data from dual port RAM to a local MegaBasic array. This data is printed to an attached terminal for display.

01-05-80 01:36 GE FANUC SERIES 90-70 DOCUMENTATION (v2.02) Page 5

<< RUNG 5 >>



<< RUNG 6 >>



| REFERENCE | NICKNAME | REFERENCE DESCRIPTION |
|-----------|----------|-----------------------|
| %Q00099 | | |
| %R00001 | | |
| %S00001 | FST_SCN | |

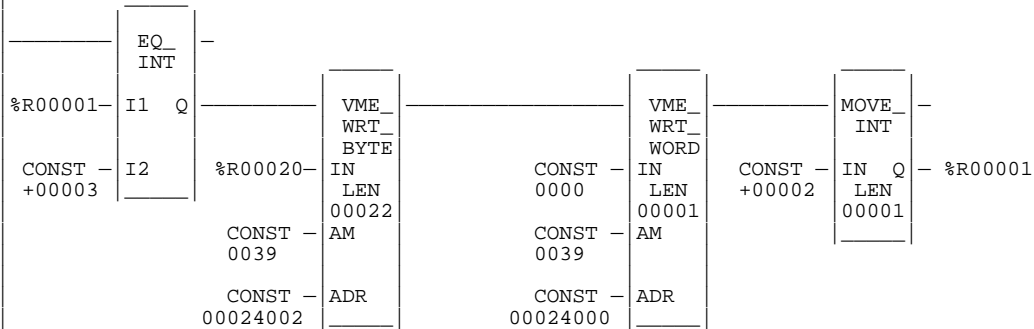
Program: PCMDemo

C:\LM90\PCMDemo

Block: _MAIN

01-05-80 01:36 GE FANUC SERIES 90-70 DOCUMENTATION (v2.02) Page 6

<< RUNG 7 >>



[END OF PROGRAM LOGIC]

| REFERENCE | NICKNAME | REFERENCE | DESCRIPTION |
|-----------|----------|-----------|-------------|
| %R00001 | | | |
| %R00020 | | | |

Program: PCMDemo

C:\LM90\PCMDemo

Block: _MAIN

```

10 Rem This program displays data from Series 90-70 PLC on attached terminal.
20 Rem
30 Access "VT100"
40 Dim integer PLC_DATA(20)           ;Rem - Array for manipulating data

60 Def integer DPR_OFFSET             ;Rem - 8086 style offset
70 Def integer FLAG_OFFSET           ;Rem - 8086 style offset
80 Def integer DPR_SEG               ;Rem - 8086 style segment

90 Rem - Clear the screen and print a banner
100 Rem
110 CLS
120 CUR 5,25
130 Print "DISPLAY OF PLC DATA VALUES"

140 Rem - Get physical address of buffer and convert to segment:offset
150 Rem
170 DPR_SEG = 0A000h
180 FLAG_OFFSET = 4000h
190 DPR_OFFSET = FLAG_OFFSET + 2

200 Rem - Set flag to start transfers and start processing data
210 Rem
220 Fill DPR_SEG:FLAG_OFFSET, @1
230 Repeat
240   While exam(DPR_SEG:FLAG_OFFSET) = 1; Next           ;Rem - Check flag
250   For I = 0 to 20
260     Exam DPR_SEG:DPR_OFFSET+I*2, @PLC_DATA(I)         ;Rem - Copy data
270   Next I
280   Fill DPR_SEG:FLAG_OFFSET, @1                         ;Rem - Clear flag
290   For I = 0 to 10                                     ;Rem - Print data
300     CUR I+8, 27
310     Print "REGISTER ", %"5I5", I+20, ":", %"5I5", PLC_DATA(I)
320   Next
330 Next

```

Section 13: Optimizing Backplane Communication

Backplane Processing for the Series 90-70 PCM

The backplane communication channel between the Series 90-70 PCM and the PLC CPU consists of a high speed parallel VMEbus using a 16 bit wide data path. On the PCM side, there are two queues for messages from the PCM to the PLC CPU and two queues for messages from the CPU to the PCM. The four queues may contain as many as 32 messages; 16 going in each direction. MegaBasic **SYSREAD** and **SYSWRITE** statements, as well as many of the MegaBasic utility packages described in this chapter, send service request messages to the PLC CPU.

Messages from the PCM to the PLC CPU are processed during the PLC system communication window, which opens once per PLC program execution sweep. If the system window is configured for run to completion mode, the default, the CPU processes all of the requests in the PCM's two outgoing queues during each sweep. As each message is processed, a response message is sent back to the PCM.

Messages from the CPU to the PCM include both responses to PCM messages and the messages sent by PLC COMMREQ function blocks. Incoming messages are generally processed immediately by the PCM.

If a PCM user program sends a large number of NOWAIT request messages in a short time (by executing **SYSREAD** and **SYSWRITE** statements, for example), the outgoing message queue may fill faster than the PLC CPU can empty it. The queues may fill even faster when CCM and a user program are both running. Eventually, one or both of the queues may fill up. When a queue is full, a new message for that queue is delayed until the PLC CPU can make room for it by servicing a message already in the queue.

In order to guarantee that every message is transferred to the PLC CPU on the next sweep, the total number of pending messages must not exceed the capacity of the PCM's backplane queue. No more than 8 low priority messages plus 8 high priority messages may be pending at any time. In addition, the system communications window must be configured for run to completion mode, or, if it is configured for limited mode, the window time limit must be long enough to service the queues.

Repeated Transfers

When a MegaBasic variable is transferred repeatedly, a timer is used to transfer it automatically at a specified interval. If a repeat value of zero is used, the variable is retransferred immediately after the previous transfer completes. This can be used to transfer data on every sweep, provided that the following requirements are met:

- The PCM is configured for MegaBasic only.
- Less than eight variables are being transferred.
- Serial port communication is light.
- There is only one PCM in the system, and it is the only intelligent option board.
- The PLC System Communications Window time is at least 10 milliseconds.
- MegaBasic does not wait for long periods of time.

Any deviation from these conditions may prevent data from being transferred on every sweep.

MegaBasic programs wait for file server activity, input on the serial ports, backplane commands issued without the **NOWAIT** argument, and the **WAIT** statement. During these times, MegaBasic variables are not updated. However, variables that are periodically read from the PLC CPU continue to be transferred into temporary buffers so that, when the program finishes waiting, it gets the latest copies of CPU variables. Periodic writes are suspended while MegaBasic is waiting.

Backplane Processing for the Series 90-30 PCM

The communication channel between the Series 90-30 PCM and the PLC CPU consists of a high speed serial backplane link. On the PCM side, there is one queue for outgoing messages from the PLC and one queue for incoming messages to the PCM. These queues may each contain two messages each.

On average, Series 90-30 PLCs can process one message or less per sweep. If CCM and the user program attempt to send several **NOWAIT** messages at once, the outgoing message queue may run out of space very quickly. If there are two or more PCMs in a Series 90-30 system, or if a PCM shares a system with other intelligent modules, The queue can fill up even more quickly. When the queue is full, the PCM allocates a buffer in memory for each additional message. Eventually, the PCM may run out of free memory.

Messages from the CPU to the PCM, including both responses to PCM messages and the messages sent by PLC COMMREQ function blocks, are generally processed immediately by the PCM.

The handling of repeated **SYSREAD** and **SYSWRITE** requests works as described above for the Series 90-70 PCM (see "Repeated Transfers").

Chapter 6

Troubleshooting Guide

This chapter provides procedures for diagnosing PCM operational problems. If the procedure given does not lead to a diagnosis, or if a problem is encountered that is not covered here, contact the GE Fanuc Series 90 Hotline (1-800-GEFANUC) for assistance.

“OK” LED Not On

1. Confirm that power is being supplied to the I/O rack containing the PCM. Verify that the PLC CPU “BD OK” LED is on. Reseat the PCM in the rack.
2. Turn the power off and then back on. Initiate a hard reset of the PCM by pressing the Restart/Reset pushbutton continuously for 10 seconds. Connect PCOP or TERMF. If the Ready prompt is displayed (TERMF) or the PCOP screen indicates that it is online, the PCM “BD OK” LED is burned out.
3. Turn the power off again, disconnect the battery cable from the connector on the circuit board, and then short the two pins on the circuit board connector. This clears PCM memory. Reconnect the battery and turn the power on again. Initiate a hard reset by pressing the Restart/Reset pushbutton continuously for 10 seconds. Check to see if the light goes on. Also, check to see if PCOP or TERMF is communicating with the PCM.
4. If the BD OK LED still is not lit and PCOP or TERMF does not respond or go online, the board may need to be returned for repair. Check the PLC fault table for a “Bad or missing module” fault. If there is a fault, return the board for repair. If there is no fault, contact the GE Fanuc Hotline for assistance.

Reset Blinks User LED1 or LED2

If the User LEDs on the PCM blink (alternating between LED1 and LED2) each time the Restart/Reset pushbutton is pressed, the PCM has not completed its power-up diagnostics. This condition occurs when the PLC CPU firmware or Logicmaster 90 software used to configure the PLC is out of date with respect to the PCM. The model number of the PCM is not recognized by the PLC or Logicmaster 90 configuration data. Upgrade both the PLC CPU firmware and Logicmaster 90 software. If both the PLC CPU and Logicmaster 90 software are up-to-date and the User LEDs on the PCM continue to blink, the PCM is at fault and should be returned for repair.

Communication Failure

General

1. Verify that both the PCM and the programmer are using the same baud rate, parity, number of data bits, number of stop bits, and the same type of handshaking (**HARDWARE**, **SOFTWARE**, or **NONE**).
2. Verify that the cable connections, described in appendix A, *PCM Cabling Information*, are correct and that the cable is firmly secured at both ends.
3. The middle light on the PCM should blink. If it does not, remove the connector from the PCM, jumper pins 4 and 5 on the PCM with a paper clip, and press the Reset/Restart pushbutton again for 10 seconds. If the LED still does not blink at least once, there is a problem with the PCM. Otherwise, the cable, programmer configuration, or programmer hardware is the problem.
4. Reconnect the cable to the PCM. If the programmer has more than one serial port, be sure the cable is connected to COM1. Set the programmer serial port to the PCM default settings. To do this when using a computer as the programmer, type **TERMF** **DEFAULT.DAT** or **PCOP** **DEFAULT.DAT**, as appropriate, at the MS-DOS prompt and press the Enter key.
5. Press and hold the Restart/Reset pushbutton for 10 seconds to initialize the PCM to its factory default settings.
6. Press the programmer Enter key while watching the USER1 LED for serial port 1 or USER2 LED for serial port 2. Each time the key is pressed the LED should blink. If the PCM has been configured by Logicmaster 90 in **BASIC** or **BAS/CCM** mode, the "Ready" prompt should also be repeated on the programmer screen; otherwise the ">" prompt should appear. If the LED does not blink or the "Ready" or ">" prompt is not displayed, either the connection from the programmer to the PCM is bad or the programmer hardware is defective.
7. Cycle power on the programmer to make sure its serial port hardware is fully reset. Problems with the programmer are very rare. When they do occur, they can often be fixed with a power cycle. If your programmer is a computer, type **TERMF** **DEFAULT.DAT** or **PCOP** **DEFAULT.DAT** again. If the LED still does not blink when a key is pressed, there is a problem with the cable or the programmer serial port hardware.

PCOP Does Not Go Online/MegaBasic Application Does Not Run

1. If you are not using PCOP, skip to step 4 below. If PCOP does not go online when the PCM is running and port 1 has been assigned through configuration to MegaBasic or CCM, initiate a hard reset by pressing the Restart/Reset pushbutton continuously for 10 seconds.
2. If the PCOP screen had gone blank or was displaying data from a MegaBasic program, press ALT-Z. A PCOP screen should be displayed, and PCOP will go online after a delay of up to 10 seconds.
3. If PCOP continues to go to TERMF after the hard reset and you have version 2.02 of PCOP, you should upgrade PCOP. To work around this problem, disconnect the PCM cable from the PCM and press ALT-Z again. Wait for PCOP to display **NO COMM**. Use the Enter key to get PCOP past the initial banner page; then, reconnect the PCM cable.

4. Verify that both the PCM and the programmer port are using the same serial port setup, especially the same type of handshaking (hardware, software, or none). If they are not using the same serial port setup, operation of the serial ports is unpredictable.
5. This should first be checked in the Logicmaster 90 software. Also, check the PCM configuration mode. If **PROG** or **BASIC** mode is selected in one of the ports, this is the serial setup used in **FACTORY** mode. Either use a different mode to determine the problem, set the serial settings to the factory default settings, or create a new **TERM.DAT** file for this setup. Repeat the previous step, using the new **TERM.DAT** file.
6. If the configuration mode is not **PROG** or **BASIC** for the port in question, the default settings are used in **FACTORY** mode. If you have not already done so, initiate a hard reset by pressing the Restart/Reset pushbutton continuously for 10 seconds.
7. If you are using PCOP, check the PCOP configuration using the configuration editor. If you are online, select Advanced functions (F10) and then Read UCDF (F8). Refer to the port screen for the serial setup for this port. Does the serial configuration match the required serial setup for the application? If not, change the PCOP configuration on this screen or reconfigure the attached device to match.
8. Type **PCOP** or **TERMF** at the MS-DOS prompt. If you still encounter communications problems between PCOP or TERMF and the PCM, the problem may be with PCOP or TERMF. Press CTRL-BREAK to return to MS-DOS. If this does not work, press CTRL-ALT-DELETE to reboot the programmer. Then, start up the PCOP or TERMF software again. If PCOP still does not go online or TERMF problems remain, a configuration mismatch probably exists between the PC and the PCM (see above).

MegaBasic Application Appears Not to Run

Refer to the errors described under the headings, “Backplane Transfer Failure,” “Insufficient Memory,” “MegaBasic Data Size,” and “Soft Reset.” An error may not be reported when the program is started on a soft reset.

PLC Fault Table Entries

1. If the PLC fault table indicates that the PCM has been lost (e.g., loss of module), check the PCM OK LED. If it is not lit, refer to the steps described in the beginning of this chapter. If the problem continues, it may be an application programming error setting the PCM watchdog timer off.
2. If the PLC fault table indicates that the rack and slot containing the PCM has an “Addition of module” fault, the PCM has not been properly configured for that slot. Check the Logicmaster 90 configuration and location of the PCM.
3. If the PLC fault table has any other fault for the PCM rack and slot, such as one of the COMMREQ faults, the problem is related to your application.
4. The PLC fault entry “Unsupported feature in configuration” indicates the PCM has firmware version 2.04 or lower and was configured for **BASIC** or **BAS/CCM** mode using Logicmaster 90 software. Change the configured mode to **PROG PRT** or **PROG/CCM**.

Backplane Transfer Failure

1. When MegaBasic aborts with a “Backplane Transfer Failure” error, it is indicating that the PLC is not communicating with the PCM. If communication with the PLC has never been established (i.e., this is shortly after starting the program and no data has been successfully moved to or from the PLC), check the PLC configuration for the PCM slot. If the PLC is configured for another module in this slot or no module at all, the PCM cannot communicate with the PLC. Reconfigure the PLC, or move the PCM to the correct slot.
2. If this is a Series 90-70 system, check that there are no empty slots to the left of the PCM (i.e., between the PLC and the PCM or between the BRM and the PCM). Blank slots prevent the PCM from communicating with the PLC and must be eliminated. After eliminating the blank slots, it is necessary to cycle power on the PLC, even if the PCM is in a remote rack.

Note

This condition does not occur with the Series 90-30 system, where empty slots are permitted.

3. If communication had been occurring with the PLC, data had been transferred and the application was running, contact the GE Fanuc Hotline for assistance. For version 1.04 or earlier of a Series 90-30 PLC Model 331, you may need to upgrade your PLC to the latest version in order to correct the problem.

Insufficient Memory Error

When this message is received, the PCM does not have enough available RAM in a large enough block to execute the requested operation. It is possible that memory has simply been fragmented through excessive use of the RAM Disk or other memory. Initiate a soft reset of the PCM by pressing the Restart/Reset pushbutton for less than 5 seconds. Then, try the operation again.

Caution

If the operation that received the original message was to save a MegaBasic program from within the MegaBasic interpreter, save the program to your hard disk by typing `SAVE PC:filename` to ensure that your work is not lost.

Too Many Files/RAM Disk Overflow

1. If files/modules have been saved to the RAM Disk through the life of this PCM, or if the PCM has been used for some other purpose before this application, the RAM Disk may be too full to permit the desired operation. Make sure that all files in the RAM Disk have been backed up by saving them to your computer's hard disk. Then, clear the PCM by using PCOP utility functions, or by disconnecting the battery and shorting the battery terminal pins on the circuit board connector while rack power to the PCM is off.
2. If you are using PCOP, check the PCM folder for unnecessary or unused files, which could be deleted or moved to another directory on the hard disk of the programmer. Then, load the PCM folder to the PCM and attempt the operation again.

If you are using TERMF, reload **BASIC.PGM** and other application files to the RAM Disk and attempt the operation again.

MegaBasic Program Save (Versions Earlier Than 2.50)

1. If the PCM is in **FACTORY** mode (a hard reset has been performed or no user configuration information is available), MegaBasic occupies almost all of the available memory in the PCM. A save to the RAM Disk in this situation could cause an insufficient memory error, since not enough room has been reserved for a RAM Disk.
2. Save the program to the PC hard disk by typing **SAVE PC:filename** to ensure that your work is not lost. You may load the program later by using the PCOP Utility functions or by typing **LOAD PC:filename** in MegaBasic.
3. Exit MegaBasic by typing **BYE**.
4. Configure the PCM for BASIC using PCOP. Take note of the MegaBasic RAM size assigned to MegaBasic but do not change this value (at least initially). If program development is still in process, do not select to start the program on reset; this is selected later when program development is more complete.
5. Load the configuration to the PCM.
6. Initiate a soft reset of the PCM by pressing the Restart/Reset pushbutton for less than 5 seconds. Re-enter MegaBasic, if desired, and continue with program development. If a start on reset had been selected, use CTRL-C to interrupt the program and continue development/debug.

MegaBasic Data Size

The PCM defaults the MegaBasic workspace size to the values given in table 4-1 in chapter 4, *MegaBasic*. If an insufficient memory error is reported on an attempt to load a program to the MegaBasic workspace, you may need to increase the RAM allocated to the MegaBasic workspace, either by upgrading the memory board (Series 90-70 PCM) or PCM model (Series 90-30 PCM), or by changing the allocation setting. The setting can be adjusted either by editing the **PCMEXEC.BAT** file (see appendix C, *PCM Commands*) or by using the PCOP utility.

An alternative solution if the program is too large for the RAM size available on the PCM is to **CRUNCH** the program. This saves program storage space in the PCM RAM Disk. Refer to chapter 4, *MegaBasic*, for information on **CRUNCH.EXE**.

Note

Crunching removes comments and extra white space. Be sure to keep a copy of the original program for documentation and maintenance purposes.

Run Mode Errors

If an application does not run when the PCM is placed in **RUN** mode with a soft reset, it may be due to an insufficient memory error. Place the PCM in **PROGRAM** mode by pushing the Restart/Reset pushbutton for 10 seconds, manually starting MegaBasic, and manually run the program to see additional error messages reported by MegaBasic.

Loss of Characters/MegaBasic Tx/Rx Failure

1. For CCM, refer to the information following this section. For MegaBasic applications, first check the serial port configuration, as previously described. The flow control characteristics on both the PCM and attached devices are particularly important.
2. In PCM firmware version 2.04, when a MegaBasic program receives characters on its STDIN channel (serial port 1 by default), input characters will be lost if a CTRL-C character is received unless CTRL-C checking is disabled. This MegaBasic statement disables CTRL-C checking:

```
xxx PARAM(1) = 1
```

Note that CTRL-C checking is usually disabled when program development is completed, regardless of the PCM firmware version. Otherwise, program execution will stop when a CTRL-C is received.

3. If you have Release 2.02 or earlier of the Series 90-70 PCM and are using the **INCHR\$** function on port 1 of the PCM, a problem exists in the PCM firmware. An upgrade to version 2.04 or later should correct the problem. Meanwhile, you may use port 2, if available, or the **NOWAIT READ** function to work around the problem.

Note

This problem does not exist in the 90-30 PCM.

4. Another possible problem could be that the type ahead buffer on the PCM has overflowed. The type ahead buffer can hold up to about 320 characters. You may want to clear the type ahead buffer periodically in your program by adding this statement:

```
xxx WHILE LEN (INCHR$ (<dev num>, 50, "", 0)); NEXT
```

5. If you have unsuccessfully tried all of these suggestions, an application error may be the problem. If you still cannot determine the cause of the problem after debug, you may need to send a copy of your program to the GE Fanuc Hotline.

CCM Data Tx/Rx Failure

1. Verify that the Logicmaster 90 or PCOP configuration for the PCM port specifies CCM, with the correct serial settings and CCM characteristics. If not, correct the specifications or change the device.
2. Confirm that the PCM is in **RUN** mode ; that is, it was powered up or placed in **RUN** mode by a soft reset. If the PCM was hard reset most recently, CCM will not function (an ACK “> ” is returned). To begin CCM communications, initiate a soft reset of the PCM.
3. If this is an RS-485 network, ensure that there are no RS-422 devices on the network (check converter boxes, amplifiers, repeaters, etc.). If there are RS-422 devices on the network, you must derate the link to an RS-422 network. Are all RS-485 lines properly terminated? Is the final node terminated? Are the cables properly made? Do they contain enough lines for full duplex operation? (See appendix A, *PCM Cabling Information*.) Are signal ground voltage differences reasonable? Are signal pairs twisted together in the cable?
4. For a Series 90-70 PLC system, check that there are no empty slots to the left of the PCM (e.g., between the PLC CPU and the PCM, or the BRM and the PCM). Blank slots prevent the PCM from communicating with the CPU or BRM and must be eliminated. This is not true of the Series 90-30 PLC system, where empty slots are permitted.
5. If the PCM is a master on the network or initiating peer, has the PLC ladder programming (COMMREQ) been done? Is the PLC CPU in **RUN** mode? If so, have you checked the status word return from the CCM COMMREQ? (See chapter 3, *CCM Operation*.) If you see the entry “other module software” faults in the PLC fault table, ensure that each COMMREQ is not enabled until the previous COMMREQ has completed.
6. If the PCM is a slave or non-initiating peer, is the Port LED (USR1 or USR2) blinking when the enquiry is sent from the initiating side? If not, check for a communications failure, as described above. Is the initiating device properly programmed?
7. If some CCM communication is obtained, but a failure is occurring during part of the communication, try selecting a different set of timeouts/retries/td value in Logicmaster 90 software or PCOP, or fine-tune the values using PCOP.
8. If this does not work, call the GE Fanuc Hotline.

Configuration Problems

Confirm that the PCM is in **RUN** mode; that is, it was powered up or placed in **RUN** mode by a soft reset to initiate the configuration. If the PCM was hard reset most recently, the configuration is not used (a ">" is returned to <ENQ>). To begin the configured action, initiate a soft reset of the PCM.

Series 90-30 Autoconfig

1. If you are trying to use Autoconfig with the Series 90-30 PLC and the PCM is not functioning in CCM **SLAVE** mode with the default configuration (see chapter 2, *Installing the PCM*), clear the PCM by using PCOP Utility functions or by removing the battery and shorting the battery terminal leads with a screwdriver. This is done in case the board has an old PCOP configuration that would take precedence over the autoconfig data.
2. If you have PLC revision 1.03 or earlier (IC693CPU331D or earlier), an upgrade is required for correct operation. Contact the GE Fanuc Hotline.
3. If multiple PCMs are present, especially of different catalog numbers (such as an ADC, CMM, or PCM301), you must configure the PCM using Logicmaster 90 software or upgrade to Release 2.0 PLC.

Logicmaster 90 Configuration

1. If you have configured the PCM using Logicmaster 90 software and the PCM is not functioning correctly in the selected mode, clear the PCM by using PCOP utility functions or by disconnecting the battery from the circuit board connector and shorting the circuit board terminal pins with a screwdriver. This is done in case the board has an old PCOP configuration that would take precedence over the Logicmaster 90 configuration data.
2. Ensure that the correct mode and setting have been chosen. See chapter 2, *Installing the PCM*, for details on configuration modes and associated parameters.
3. If multiple PCMs are present, especially of different catalog numbers (such as an ADC, CMM, or PCM301), you may need to upgrade to Release 2.0 PLC in order to configure the PCM.
4. PCMs with firmware version 2.04 or earlier do not support the Logicmaster **BASIC** or **BAS/CCM** configuration modes.

PCOP Configuration

1. Using the configuration editor, verify that the configuration specifies the desired operation. Did you load the configuration data to the PCM? In **ONLINE** mode, press MDIR from the Utility screen. A file called UCDF should be present on the PCM.
2. Edit UCDF (from the Advanced functions, press READ UCDF) to make sure the file contains the correct configuration.
3. For CCM configuration, is CCM enabled on this port? Refer to the procedure above, under the heading, “CCMTx/RxFailure.”
4. For MegaBasic configuration, is MegaBasic enabled on the desired port? Are STDIN, STDOUT, and STDERR set to the desired port or other location? Have you selected to start the program automatically from reset and entered the program name? (This is usually desired.)
5. If the configuration does not take effect, check the PCM Runtime Errors screen. Possible configuration errors, which might be displayed on this page, include “Insufficient Memory” and “Module not Found.”

PCOP Screen Goes Blank or PCOP Locks Up

1. Check that there are no drivers or other packages (e.g., communication or networking) loaded on the PC by the **CONFIG.SYS** or **AUTOEXEC.BAT**. Serial port and high memory usage packages are especially a problem.
2. In PCOP Release 2.02, a known bug existed on some PCs that would cause PCOP to jump to the TERMF page immediately. If this happens, you can work around it by disconnecting the PC-to-PCM cable at one end, proceeding into PCOP beyond the Folder Select screen, and then attaching the cable. An upgrade to PCOP is recommended.
3. If the PCM is running a MegaBasic or CCM application, PCOP may detect characters on the serial port and switch to TERMF. Initiate a hard reset and then press ALT-Z to return to PCOP.
4. If the screen is either completely blank or the menus are incorrectly displayed, use the **TERMSET** utility function to verify that the proper monitor is selected in **TERM.DAT**. The default display adapter and monitor is set for a Workmaster or Workmaster II industrial computer, CGA driving a monochrome monitor. The default may also be obtained by using **DEFAULT.DAT**.
5. Make sure that there is no other equipment, other than a PCM, attached to the serial port being used by PCOP (COM1 by default). If a foreign device is detected, PCOP jumps to TERMF. Pressing ALT-Z usually flickers the screen and returns to TERMF. Disconnect the other device and connect the cable to a PCM or work offline.

Appendix A

PCM Cabling Information

This appendix provides cabling specifications and wiring diagrams for the Series 90 PCM.

Cable and Connector Specifications

- Cable connector to PCM Ports 1 or 2: Male, Subminiature-D Type, Cannon DB25P (solder pin) with DB110963-3 Hood; AMP shell 207345-1 and connector 205208-1 with crimp pin, 66506-1, or solder pin, 66570-3; or equivalent standard RS-232C connector.
- Connectors for ports 1 and 2 are located on the Series 90-70 PCM board. Both ports are brought out a single connector on the Series 90-30 PCM board. A Wye cable is provided with each Series 90-30 PCM board.
- Length (maximum):
 - 50 feet (15 meters) for RS-232C.
 - 50 feet (15 meters) for RS-422/RS-485 without isolation at the remote end.
 - 4000 feet (1200 meters) for S-422/RS-485 with isolation at the remote end.
- Overall shield.
- 24 AWG (minimum).
- Connector to external device, specified by external device manufacturer.

The following cables provide acceptable operation at data rates up to 19.2K BPS and distances up to 4000 feet for RS-422/RS-485.

| Belden Catalog No. | Construction | Application |
|--------------------|---|--|
| 9505 | 5 pairs #24 AWG stranded, overall shield. | RS-232, RS-422 or RS-485 at 19,200 BPS or below. |
| 9306 | 6 pairs #22 AWG solid, overall shield. | RS-232, RS-422 or RS-485 at 19,200 BPS or below. |
| 9832 | 5 pairs #24 AWG stranded, overall shield. | RS-422 up to 38,400 BPS. |
| 9731 | 6 pairs #24 AWG stranded, pairs individually shielded, low capacitance. | RS-422 up to 38,400 BPS. |
| 8105 | 5 pairs #24 AWG stranded, overall shield, low capacitance. | RS-422 up to 38,400 BPS. |
| 9844 | 4 pairs #24 AWG stranded, overall shield, low capacitance. | RS-485 up to 38,400 BPS. |

At shorter distances under 50 feet (15 meters), almost any twisted pair or shielded twisted pair cable will work, as long as the wire pairs are connected correctly. Do not use the shield as a signal ground conductor.

When using RS-422/RS-485, the twisted pairs should be matched so that both transmit signals make up one twisted pair and both receive signals make up the other twisted pair. If this is ignored, cross-talk resulting from the mismatching could affect the performance of the communications system.

When routing communication cables outdoors, transient suppression devices can be used to reduce the possibility of damage due to lightning or static discharge.

Caution

Care should be exercised to ensure that both the PCM and the device to which it is connected are grounded to a common point. Failure to do so could result in damage to the equipment.

Serial Connectors

The Series 90-70 PCM has two serial connectors; each one supports both RS-232 and RS-485 operation. The serial ports are identical, and either port can be used for most applications. The two ports are configurable for different communication parameters.

Note

The connector pin assignments for the Series 90-70 PCM are shown below:

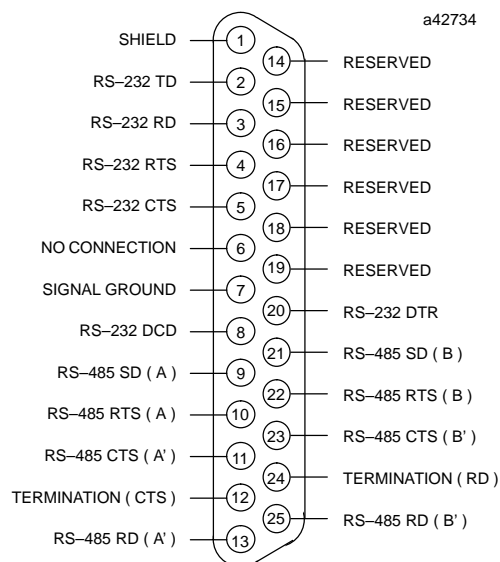


Figure A-1. Serial Port Pin Assignments for the Series 90-70 PCM

Note

In figure A-1, SD (Send Data) and RD (Receive Data) are the same as TXD and RXD (used in Series Six PLC publications). (A) and (B) are the same as – and +. A' and B' denote inputs, and A and B denote outputs. To terminate the RS-485 CTS input signal, jumper pins 11 and 12; to terminate the RD input signal, jumper pins 24 and 25.

The Series 90-30 PCM has a single serial connector that supports two ports. One port has a fixed interface. Port 1 uses RS-232 operation only. The IC693PCM300 module is restricted to using RS-485 on port 2. All other Series 90-30 PCM modules may select either RS-232 or RS-485 operation on port 2.

The connector pin assignments for the Series 90-30 PCM are shown below.

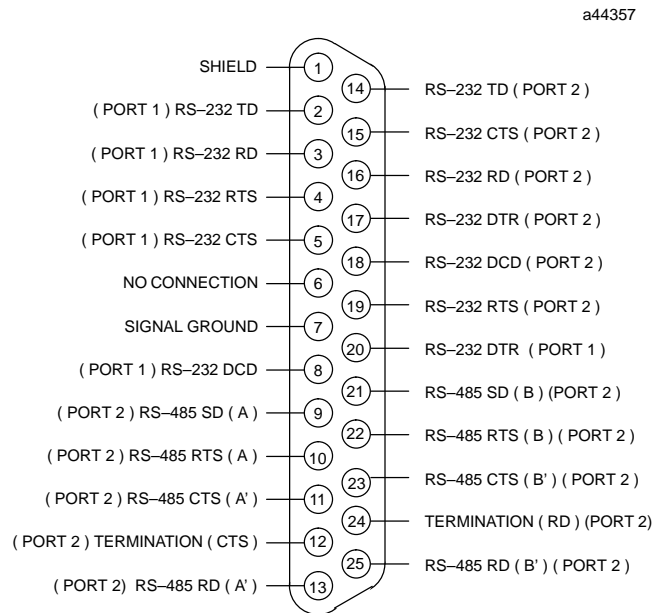


Figure A-2. Serial Port Pin Assignments for the Series 90-30 PCM

A WYE cable is supplied with each Series 90-30 PCM. The purpose of the WYE cable is to separate the two ports from a single physical connector; i.e., the cable separates the signals. In addition, the WYE cable makes cables used with the Series 90-70 PCM fully compatible with the Series 90-30 PCM.

The WYE cable is 1 foot in length and has a right angle connector on one end that connects to the PCM. On the other end, it has a dual connector with one connector for port 1 and the other for port 2.

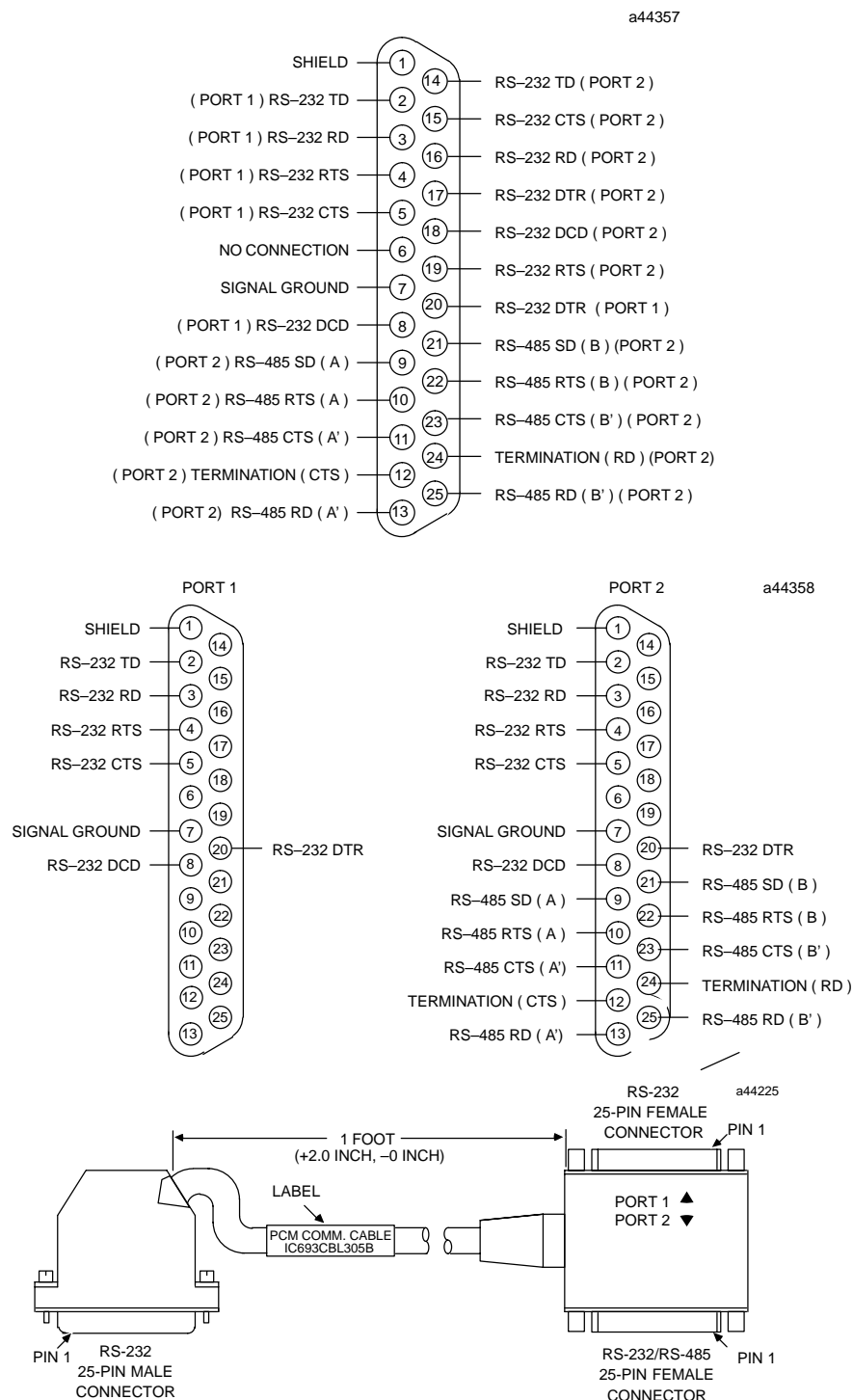


Figure A-3. WYE Cable Connections for the Series 90-30 PCM

In order to use an RS-232 cable on port 2 of the Series 90-30, either a special cable must be made according to the serial port pin assignments shown above or a WYE cable must be used. Standard Series 90-70 PCM cables can be used for the Series 90-30 PCM when the WYE cable is used.

Cabling

The prewired cables shown below provide the required signal connections between the RS-232 serial port on a PCM and a serial port on the programmer. Each of the cables physically appear the same; the only difference is the internal pin connections.

An IC697CBL701 cable provides the required signal connections between a PCM and a Workmaster industrial computer.

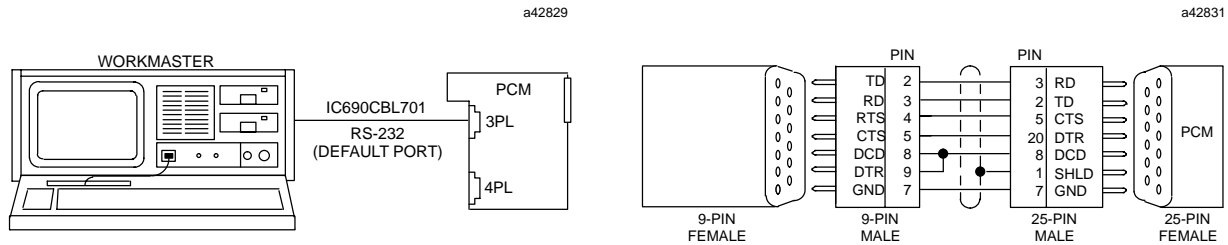


Figure A-4. PCM to Workmaster Computer

An IC697CBL702 cable provides the required signal connections between a PCM and an IBM PC-AT personal computer.

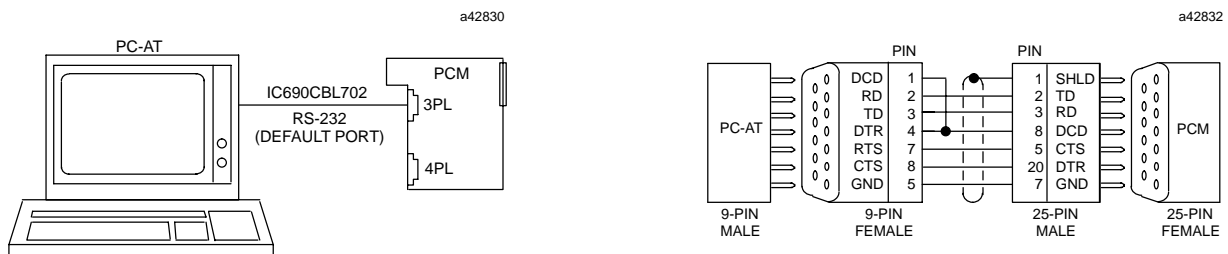


Figure A-5. PCM to PC-AT Personal Computer

The following illustration shows the connection between a PCM and a Simplicity Model W industrial computer. The IC697CBL702 cable may be used for this connection.

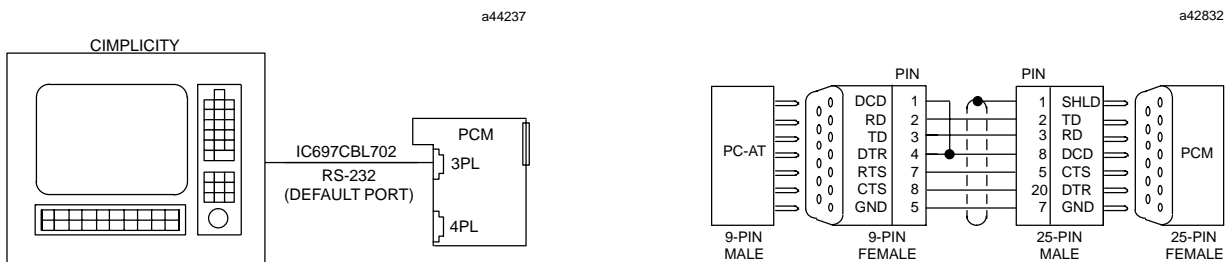


Figure A-6. PCM to Simplicity Model W Computer

An IC697CBL705 cable provides the required signal connections between a PCM and a Workmaster II industrial computer or an IBM Personal System/2 personal computer.

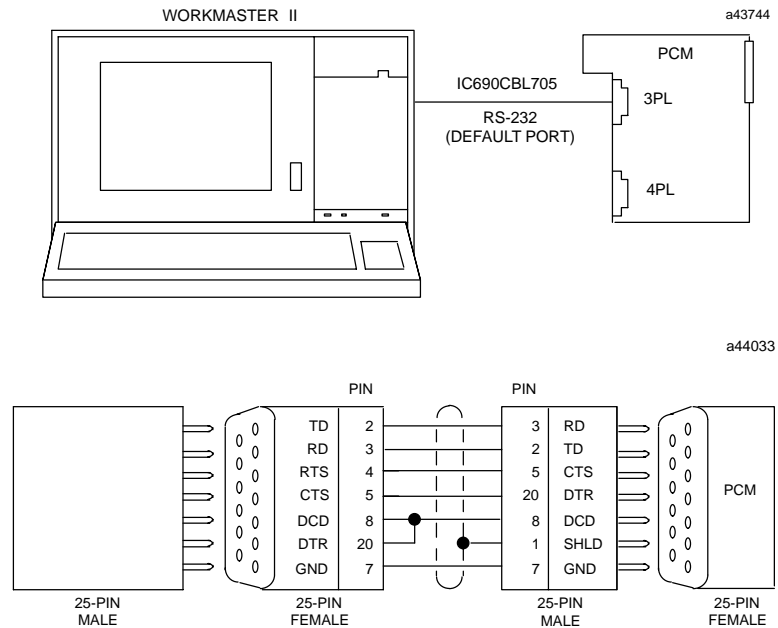


Figure A-7. PCM to Workmaster II Computer or PS/2 Computer

Connect the cable's 25-pin male connector to the top serial port female connector on the front of the PCM. Then, connect the cable's 9-pin or 25-pin female connector to the male RS-232 connector (serial port) on the selected programming device.

For more information on these cables, refer to the *Series 90-70 Programmable Controller Cables – PCM to Programmer Data Sheet*, GFK-0359.

RS-232 Cables

Typical cable wiring for many PCM RS-232 applications is shown in the following illustrations.

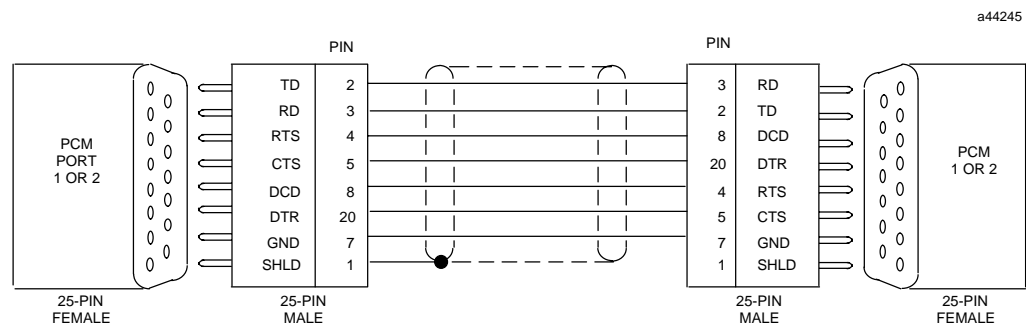


Figure A-8. PCM to PCM with Hardware Flow Control (RS-232 only)

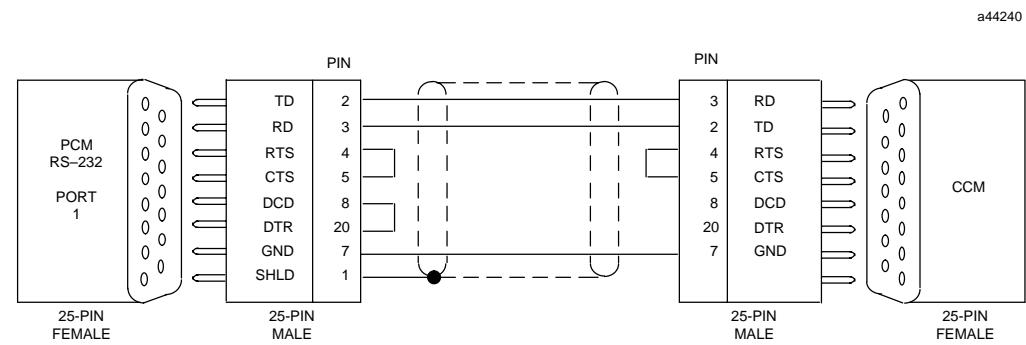


Figure A-9. CCM2 to PCM (RS-232 only)

a44234

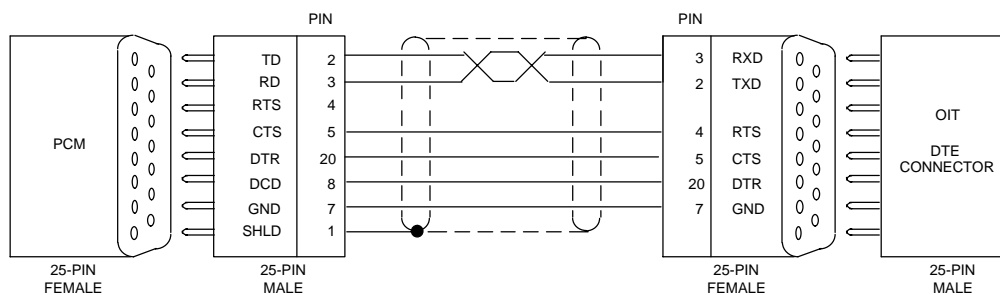


Figure A-10. PCM to OIT with Hardware Flow Control (RS-232 only)

a44239

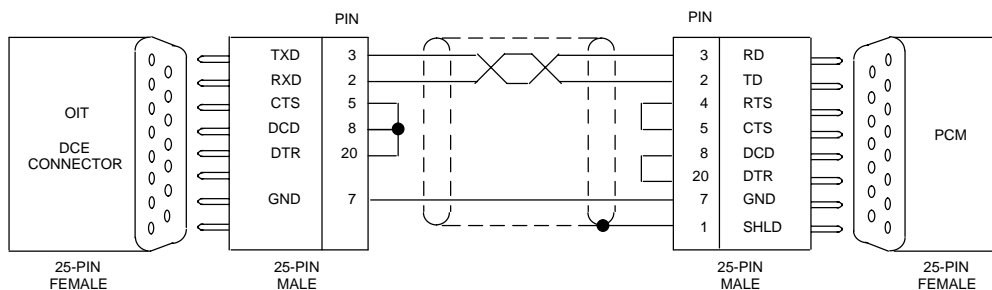


Figure A-11. PCM to OIT without Hardware Flow Control (RS-232 only)

Note

Some versions of the GE Fanuc Operator Interface Terminal (OIT) have an RS-232 DCE connector (labeled “Secondary Port”) as well as a DTE connector (labeled “Primary Port”). Other versions have only a DTE connector.

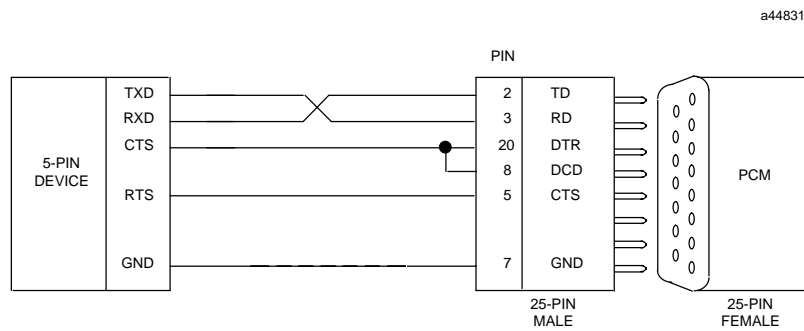


Figure A-12. PCM to a 5-Pin Device, Full Hardware Flow Control

In the following illustration, the 5-pin device cannot flow control the PCM:

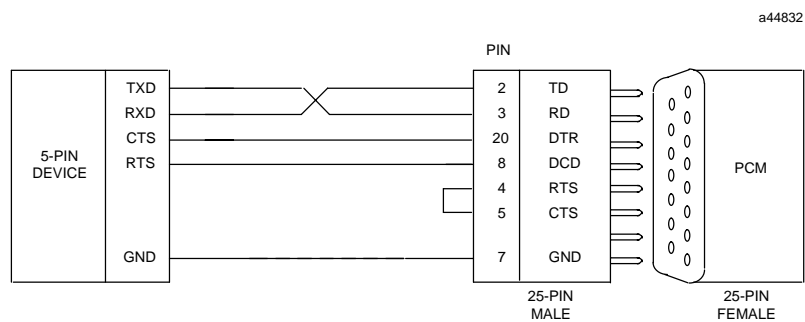


Figure A-13. PCM to a 5-Pin Device, No Flow Control or Hardware Flow Control

RS-422/RS-485 Cables

The RS-422/RS-485 signal nomenclature used in this manual can be cross-referenced to EIA standard RS-422, as shown below:

| CCM Signal Name | RS-422 Standard Signal Name |
|-----------------|-----------------------------|
| RS-422 SD(B) | B |
| RS-422SD(A) | A |
| RS-422 RD(B) | B' |
| RS-422RD(A) | A' |

During a mark state (logic 1), B is positive with respect to A. During a space state (logic 0), B is negative with respect to A.

When connecting the PCM to a non-Series 90 device using the RS-422/RS-485 standard, the non-Series 90 device's line receiver must contain "fail safe" capability. This means that in an idle, open, or shorted line condition, the output of the line receiver chip must assume the mark (logic 1) state.

When using RS-422/RS-485, the twisted pairs should both be matched so that both transmit signals make up one twisted pair and both receive signals make up another twisted pair.

The PCM is supplied with a 120 Ohm terminating resistor in each RS-422/RS-485 receiver circuit. If the module is at either end of an RS-422/RS-485 multidrop or point-to-point link, these resistors should be in the circuit. If the module is an intermediate drop on the multidrop link, the appropriate resistors should be disconnected from the circuit by removing their jumpers.

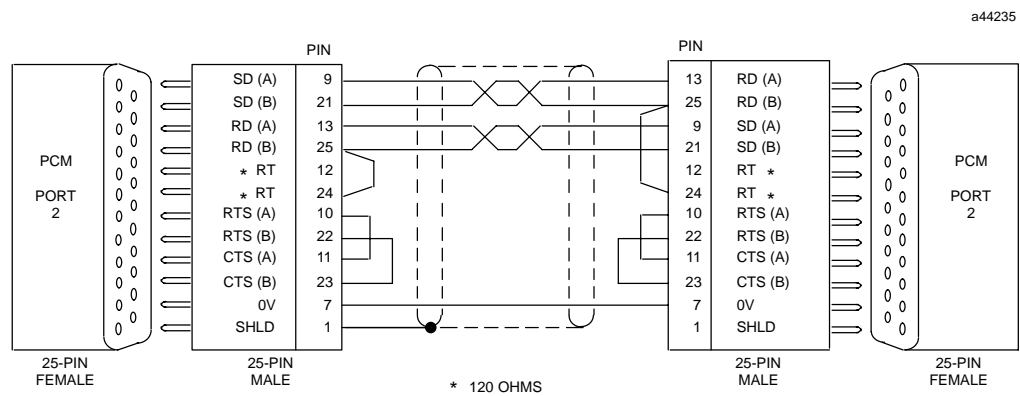


Figure A-14. PCM to PCM without Hardware Flow Control (RS-422/RS-485)

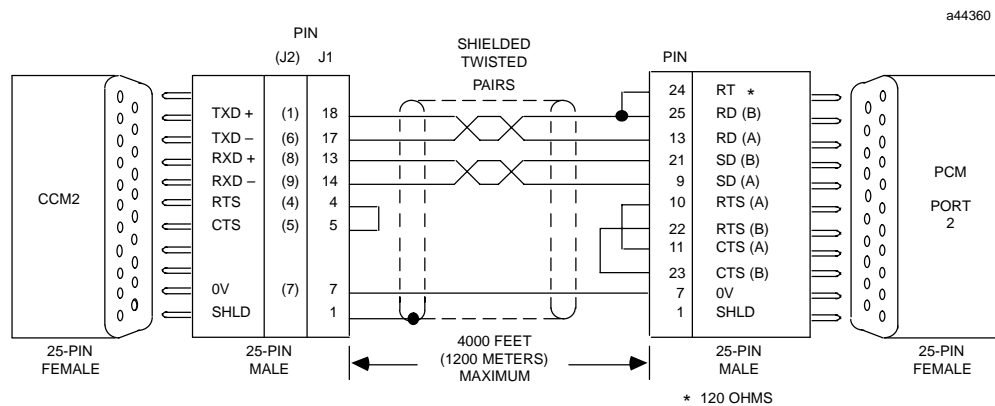


Figure A-15. CCM2 to PCM (RS-422/RS-485)

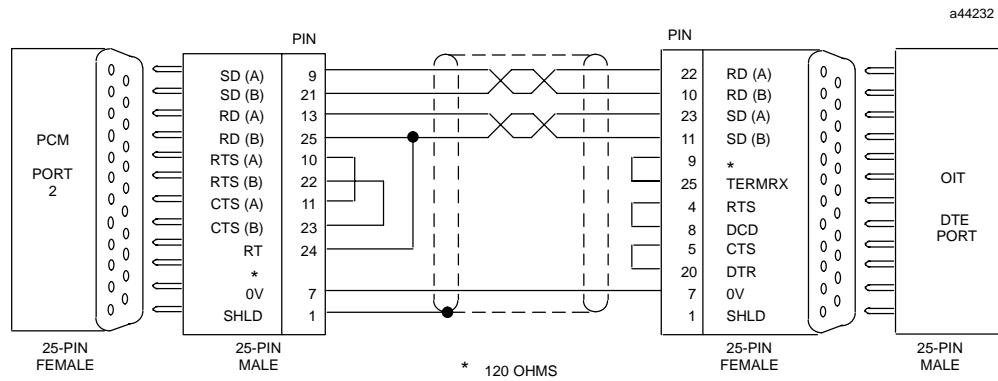


Figure A-16. PCM to OIT without Hardware Flow Control (RS-422/RS-485)

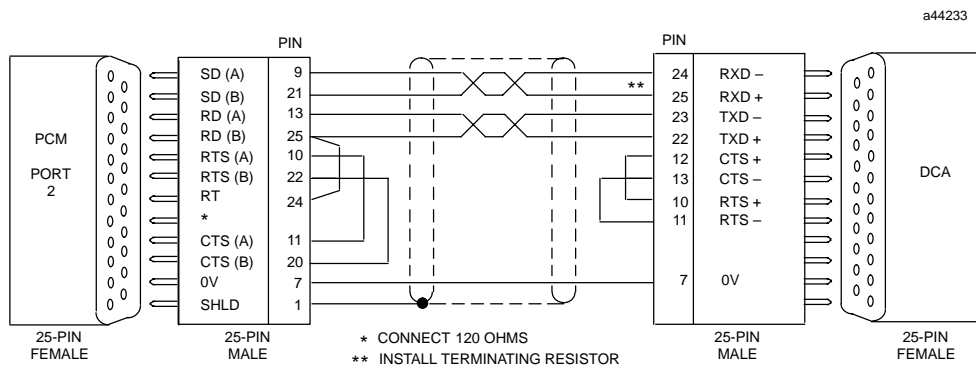


Figure A-17. PCM to Series One/Series Three DCA (RS-422/RS-485)

a45362

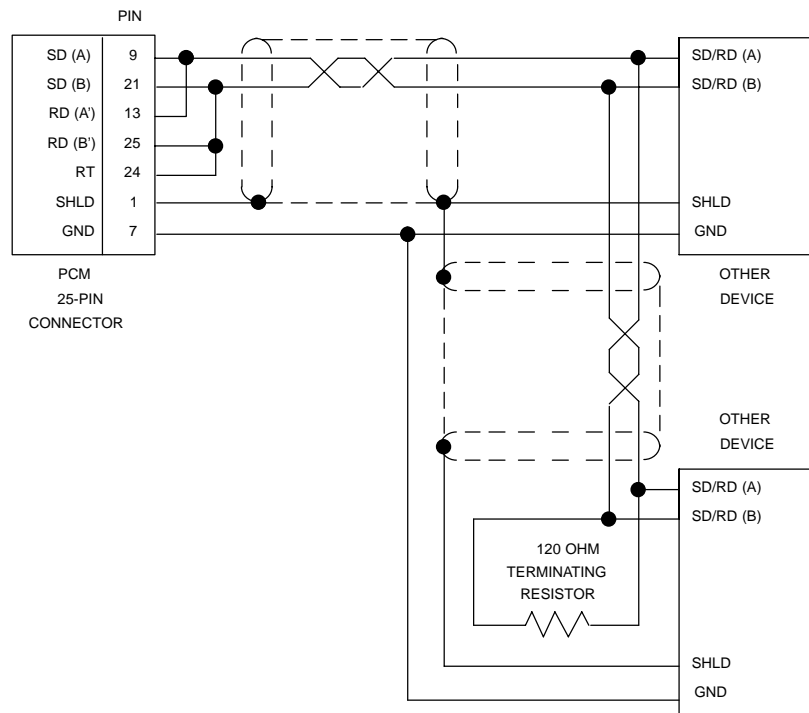


Figure A-18. 2-Wire RS-422/RS-485 PCM Hookup

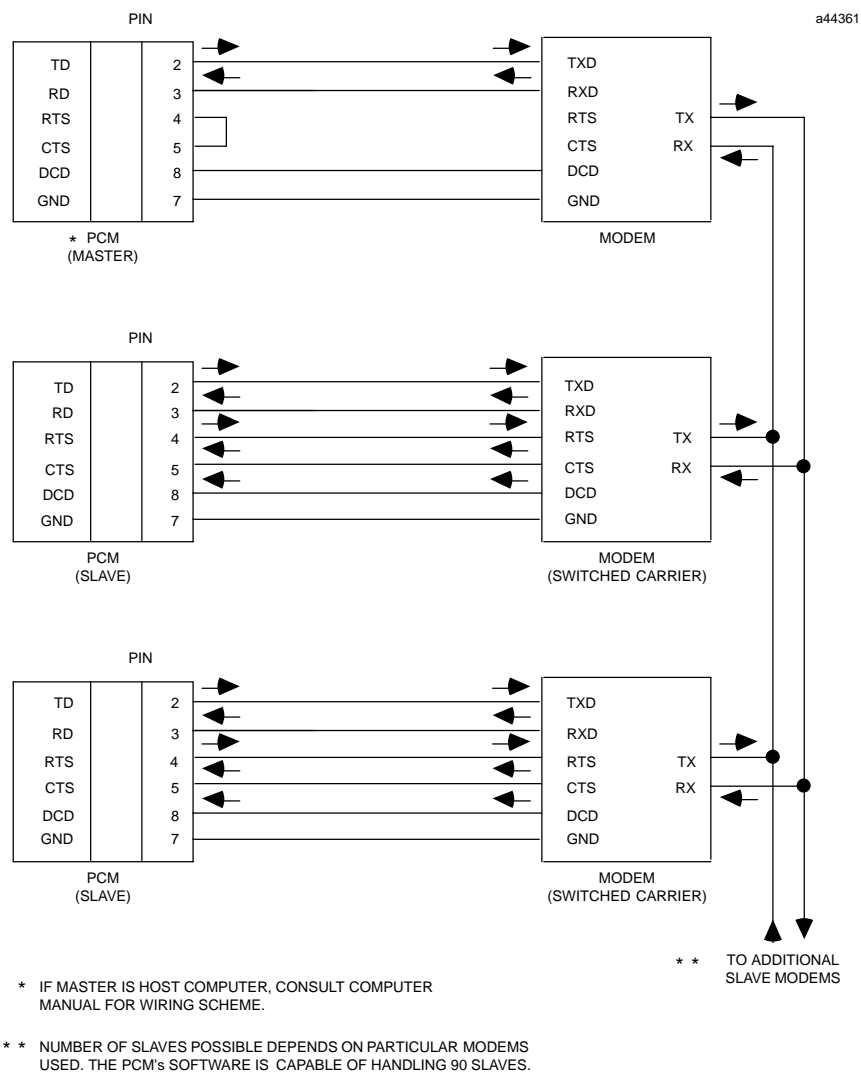


Figure A-19. CCM2 or Host Computer to Multiple PCMs (Multidrop)

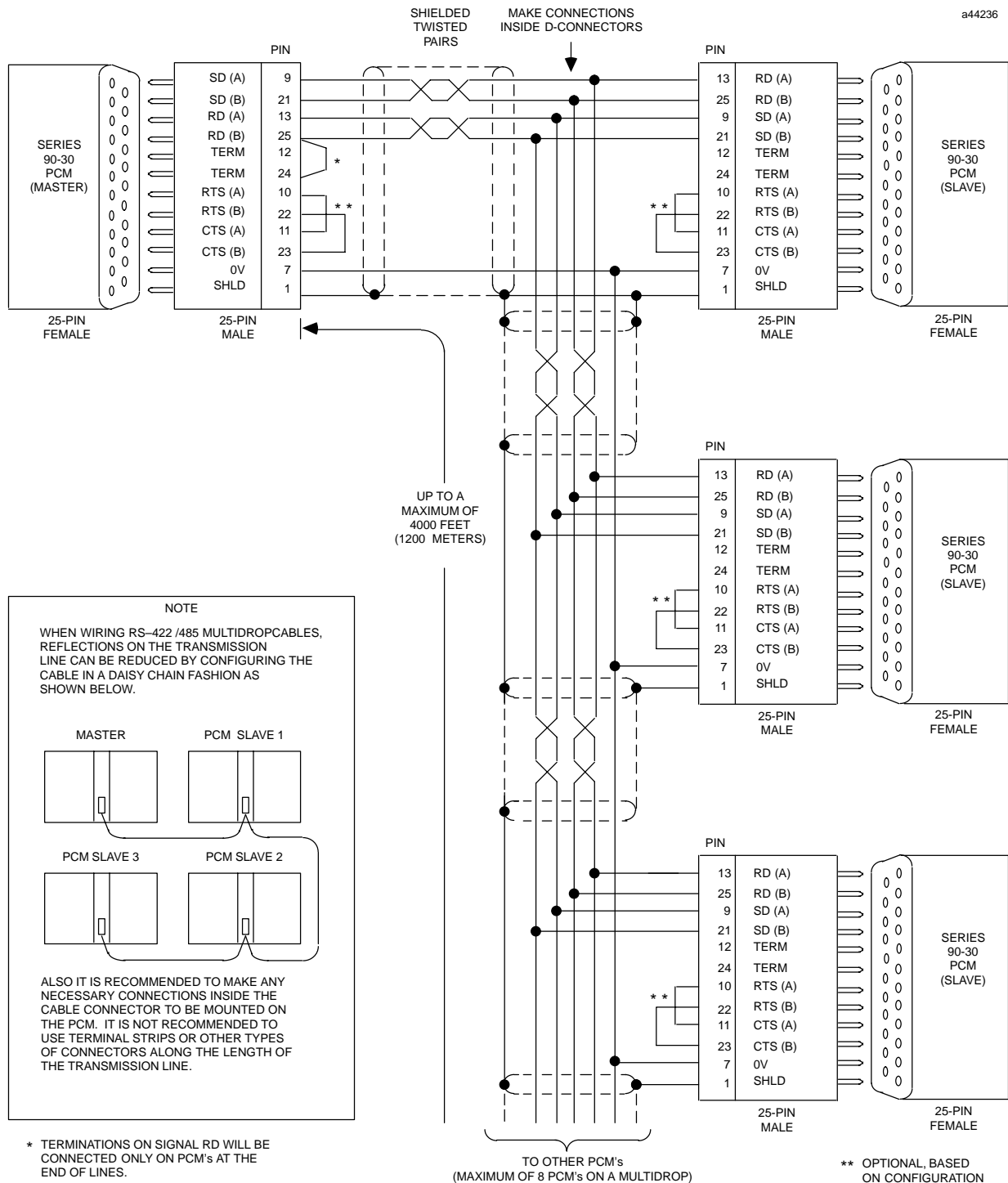


Figure A-20. PCM or Host Computer to Multiple PCs (4-Wire Multidrop)

Note

If the PCM is configured for no flow control, the jumpering of RTS/CTS is not required.

Appendix *B*

Resetting the PLC from a PCM Program

PLC programs can reset any PCM, CMM, ADC, or GDC module by sending a backplane message from a COMMREQ function block. The reset message produces exactly the same effect as switching PLC power off and on or pressing the module Restart/Reset pushbutton for less than five seconds. However, if the module watchdog timer expires, causing the OK LED to go off, the reset message will have no effect.

The COMMREQ function block SYSID parameter must be programmed to send the message to the rack and slot where the target module is installed. The TASK parameter must be 127 decimal (7F hexadecimal). The COMMREQ command/data block must be initialized as shown below. Note that there is never a return status; the COMMREQ must be sent in NOWAIT mode.

| COMMREQ Data Block Field | Value |
|------------------------------|--|
| Length | 2 |
| WAIT/NOWAITFlag | 0 (Ignored) |
| Status Pointer Memory Type | 0 (Ignored) |
| Status Pointer Memory Offset | 0 (Ignored) |
| Idle Timeout Value | 0 (Ignored) |
| Maximum Communication Time | 0 (Ignored) |
| Reset Status Value | 0 (Ignored) |
| Reset Command Value | 4320 Hexadecimal (soft reset) 8640 Hexadecimal (hard reset) |

The two Logicmaster 90-70 ladder program rungs shown below will send a soft reset to a Series 90-70 PCM whenever contact %T00001 is active and contact %T00002 is inactive. Latching contact %T00002 assures that only one reset COMMREQ is set.

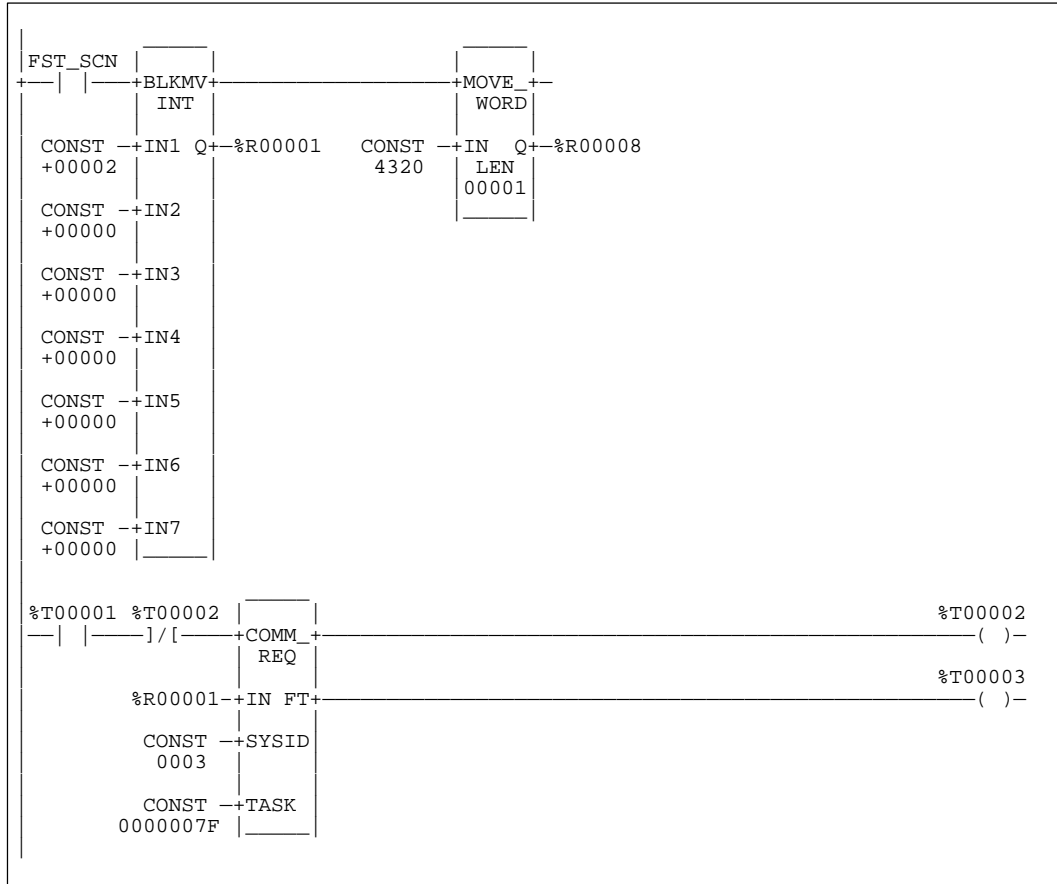


Figure B-1. Soft Reset

The next Logicmaster 90-70 program fragment will send a hard reset to a Series 90-70 PCM whenever contact %T00011 is active and latching contact %T00012 is inactive.

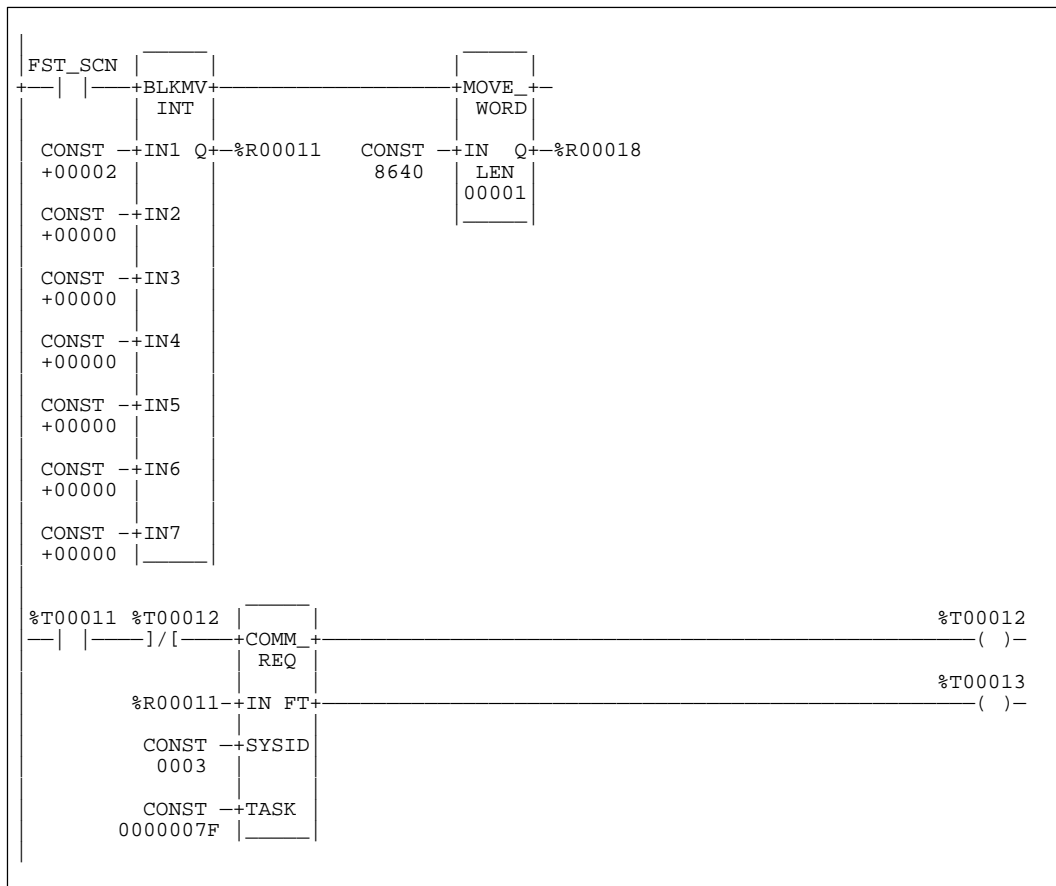


Figure B-2. Hard Reset

Logicmaster 90-30 versions of these rungs are slightly different because the Logicmaster 90-30 COMMREQ function block does not have a power flow output.

Appendix C

PCM Commands

The PCM includes a command interpreter which is similar in principle to the MS-DOS command line interpreter or UNIX shell. PCM commands provide complete control for loading and storing applications, and for executing them.

Note

Your PCM must have firmware version 2.50 or greater in order to use the commands described in this appendix.

Accessing the Command Interpreter

The PCM command interpreter is connected by default to PCM serial port 1 whenever the PCM is not configured by Logicmaster 90 in CCM ONLY mode and is not executing an application program. In addition, the command interpreter can be accessed through the PCM backplane, as described earlier in this document. The following discussion assumes that you are trying to access the command interpreter through serial port 1 using the TERMF terminal emulation program. TERMF is described fully in chapter 2, section 4, *TERMF Installation and Configuration*.

When a PCM is configured in PCM CFG mode using Logicmaster 90 software and there are no files stored in it, the command interpreter will be connected to serial port 1 whenever the PCM is reset by holding the restart button for 10 seconds (a hard reset). Pressing the Enter key displays a ">" prompt from the interpreter when it is active. Pressing the Enter key repeatedly adds another ">" prompt on the same line each time you press the Enter key.

Depending on the Logicmaster 90 configuration for the PCM, the MegaBasic interpreter may start at power-up or a reset. MegaBasic prints a startup banner message (by default to port 1) whenever it starts. When you see the startup message and a Ready prompt, you can type **BYE** and press the Enter key to exit from MegaBasic to the command interpreter. If you see the startup message but no Ready prompt, MegaBasic is running a program. You can usually stop the program by typing **CTRL-C**. (Press and hold down the CTRL key while typing C.)

If you cannot access the command interpreter after trying the procedures described above, use the Logicmaster 90 configuration software to check the PCM configuration. If a configuration has been stored to the PLC, load it to the Logicmaster software and then check the PCM configuration mode to be sure it is either BASIC, BAS/CCM or PCM CFG. If the PCM mode is not one of these, change it and store the new configuration to the PLC. If there is no PLC configuration, create one with the PCM configured to PCM CFG mode and then store it to the PLC.

As a last resort, try turning off power, disconnecting the battery cable from the connector on the circuit board, and shorting the two pins on the circuit board connector. This clears PCM memory. Reconnect the battery and turn the power on again. If there is no MegaBasic startup banner or command prompt, and you are sure the PCM configuration mode is correct, refer to chapter 6, *Troubleshooting Guide*.

Interactive Mode

When the PCM connects you to the command interpreter after power up or a reset, you should see this prompt:

->

When you return to the command interpreter from MegaBasic, you may not see a prompt on your screen immediately, but pressing the Enter key should display a ">" prompt. At this point, the interpreter is in its default mode, which is used to communicate with the PCM development software package, PCOP. Default mode does not respond to you with text messages, nor does it echo the keys you type back to your screen. You must switch to INTERACTIVE mode by typing two exclamation points (!!) and pressing the Enter key. This message will appear:

```
INTERACTIVE MODE ENTERED
type '?' for a list of commands
```

If your PCM firmware version is earlier than 3.00, you will see "DEBUG" rather than "INTERACTIVE" in this message.

To display a list of PCM commands, type a question mark (?) and press the Enter key. If you are using PCOP, you need to type three exclamation points (!!!) and then press the Enter key in order to return to it. PCOP cannot communicate with the command interpreter while it is in interactive mode.

Caution

When using Megabasic, make sure all your programs have been saved to your computer (the PC: device) before attempting to use the command interpreter.

Command Format

All PCM commands begin with a single letter which identifies the command. The complete command is an ASCII string, terminated by an ASCII CR (0D hexadecimal) character. Command arguments are separated from the command character and each other by one or more spaces.

Note

When using PCM commands in batch files with certain PCM firmware versions (see appendix D, *PCM Batch Files*), the command letter must be uppercase characters. You can avoid batch file errors by using uppercase characters exclusively in PCM commands.

Notation Conventions

Arguments are shown as symbolic names within angle brackets (< >). For example, <file_name> represents a string of ASCII characters containing the name of a file, <pcm_filename> represents a string of ASCII characters containing the name of a PCM file, <led_use_code> represents two ASCII characters containing a one-byte hexadecimal value, etc.

Optional arguments are shown in square brackets ([]). They may be omitted; all other arguments are required.

Commands

PCM commands are summarized in the following table:

| Command | Description |
|---------|---|
| L | Load a file from the PC. |
| S | Save a file to the PC. |
| D | Show a directory of files in memory. |
| X | eXterminate (delete) a file. |
| R | Run an executable file. |
| K | Kill a running task. |
| C | Clear the PCM. |
| @ | Execute a batch file. |
| F | Show available memory. |
| G | Get PCM memory ID. |
| H | Get the PCM revision number. |
| B | Configure a user LED. |
| U | Reconfigure the PCM. |
| M | Create a memory module. |
| PT | Show PCM task information. |
| PC | Show PCM config errors. |
| PM | Show reset type and mode. |
| PL | Show the location of the PCM. |
| PD | Dump the state of the PCM just before the last soft reset to a PC file. |
| !! | Enter interactive (DEBUG) mode. |
| !!! | Exit interactive (DEBUG) |

The following commands are also available, although most of these are seldom used except by PCOP.

| Command | Description |
|---------|-----------------------------|
| I | Initialize a device. |
| J | Format the ROM: device. |
| O | Get LED configuration. |
| Q | Set protection level. |
| V | Verify a file. |
| W | Wait for a background task. |
| Y | Set upper memory limit. |

The remainder of this section provides detailed descriptions of the commands listed in the preceding tables. The commands are presented in alphabetical order.

@ (execute a batch file)

Format: @<file_name>

This command executes the PCM batch file <file_name>. No intervening space is permitted between the @ command and the file name. These examples show how the @ command is used. The file extension is optional; in the last example, **MYFILE.BAT** is executed. If no device is specified, as in the first and last examples, RAM: is assumed.

```
@MY.BAT
@PC:A:\MYDIR\MY.BAT
@MYFILE
```

These errors can be returned:

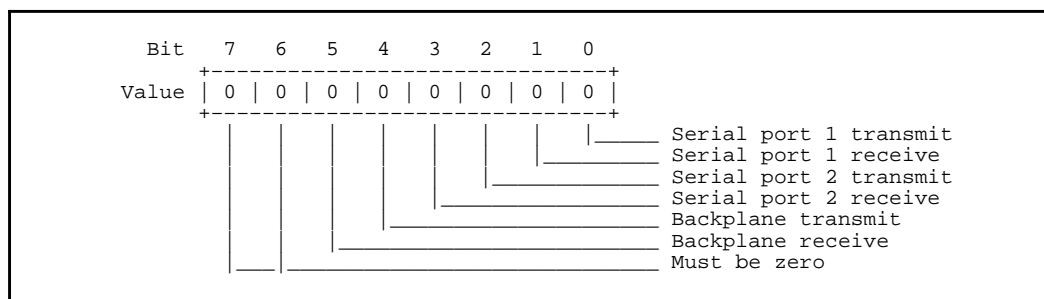
```
File not found <file_name>
Illegal module type <file_name>
```

For more information on PCM batch files, refer to appendix D, *PCM Batch Files*.

B (configure LEDs)

Format: B <led_number> <led_use_code> [<task_number>]

This command configures either of the bottom two PCM LEDs (USER1 and USER2). The LED number can be either of the ASCII characters 1 or 2. The LED use code is a two-digit ASCII hexadecimal code that specifies a configuration byte for the LED. Binary values for the LED use code are as follows:



Combine the bits for the desired LED action into a single byte value in hexadecimal format. For example, to blink LED USER1 when characters are transmitted or received on serial port 2, use binary code 00001100. This is equivalent to 0C hexadecimal, so the command is **B 1 0C**.

The task number, if specified, is an ASCII hexadecimal digit that specifies which task will control the indicated LED. The task number must be the number of a valid task (0-0F hexadecimal). When an LED is assigned to a task, the LED use code must be specified as 40 hex. To configure LED 1 to be controlled by task 7, use **B 1 40 7**.

Task 7 will then be able to change the behavior of LED 1. A second **B** command may be used specify a default communication event or events which will flash the LED before task 7 programs it.

C (Clear the PCM)

Format: C

This command deletes all RAM Disk files on the PCM, including the UCDEF, resets the PCM, and initializes it in factory mode. The **C** command returns an error if any files are in use. When this happens, the PCM must be put in factory mode (U FDEF) before it can be cleared.

D (file Directory)

Format: D [<option>]

This command prints the names of the files stored in the PCM RAM Disk or other file devices. It returns no errors. A single letter <option>, separated by a space from the D command, may be used. The following table shows the data returned by this command and its options.

| Option | Description |
|--------|---|
| D | The D command used alone shows the names of non-hidden files in the PCM RAMDisk. |
| D H | This command option shows all files, hidden and non-hidden, in the PCM RAM Disk. Hidden files may include files loaded to the PCM using the Hidden attribute as well as file data blocks appended to files created by PCM applications. |
| D P | This command option shows files in the PCM system EPROM. These files provide VTOS functionality. |
| D R | This command option shows files in the PCM option EPROM. These files provide functionality such as MegaBasic, CCM, etc. |

F (show Free memory)

Format: F

In interactive mode, this command displays the amount of free PCM memory as two decimal numbers:

```
Total available memory is xxxxxx bytes
Largest available block is yyyyyy bytes
```

The first is the total of all available memory. The second is the size of the largest free memory block. Both sizes are expressed in bytes. No errors are returned.

Note

The largest available block size is often used to determine the amount of memory to allocate to MegaBasic. The MegaBasic data space must be smaller than the largest available block size.

G (Get hardware ID)

Format: G

This command returns the ID number of the PCM hardware configuration. The value is an ASCII string containing two hexadecimal digits which specify the ID. One of the following codes is returned:

| Code | Description |
|------|--|
| 00 | Series 90-70 PCM with no daughter board. |
| 1D | Series 90-70 PCM with a 512K daughter board (640K bytes total). |
| 1E | Series 90-70 PCM with a 256K daughter board (384K bytes total). |
| 1F | Series 90-70 PCM with a 128K daughter board (256K bytes total). |
| 1C | Series 90-70 PCM with a 64K daughter board (192K bytes total). |
| FF | Series 90-30 PCM model IC693PCM300 (160K bytes). |
| FE | Series 90-30 PCM model IC693PCM301 (192K bytes). |
| FC | Series 90-30 PCM model IC693PCM311 (640K bytes). |
| 80 | Graphics Display Coprocessor Module with a video daughter board. |
| 81 | Series 90-70 Alphanumeric Display Coprocessor Module. |
| 82 | Series 90-30 Alphanumeric Display Coprocessor Module. |

H (get PCM firmware revision number)

Format: H

This command returns the firmware release number of the PCM. The ASCII string returned by the PCM contains a single digit for the major revision number, a period, and two digits for the minor revision number. For example:

```
Software revision number is 3.03
```

I (Initialize device)

Format: I <device_name> <device_initialization_string>

This command sends the specified initialization string to the specified device. Currently, the two serial devices, COM1 and COM2, and the CPU device support this command.

A space character is required between the device name and initialization string. The parameters in <device_initialization_string> must occur in the order listed in the table below with no intervening spaces. Any number of parameters may be omitted at the right end of the string. Parameters to the left of the last one may be omitted, but all the surrounding commas must be included. Omitted parameters retain their previous settings.

| Device | Initialization String |
|----------------|--|
| COM1: COM2: | <p><baud_rate>,<parity>,<data_bits>,<stop_bits>,<flow_control>,<physical_interface>,<duplex_mode>,<delay_value>,<typeahead_size></p> <p>where:</p> <p><baud_rate> = 300, 600, 1200, 2400, 4800, 9600, 19200*, or 38400 – the number of bits per second. Note that 38,400 baud is supported only by the Series 90-70 PCM, and only for RS-422 or RS-485 port configurations.</p> <p><parity> = O, E, N* - the type of parity checking: Odd, Even, or None.</p> <p><data_bits> = 7 or 8* - the number of data bits per character. Use 8 unless text with 7 bit characters will be the <i>only</i> data transferred.</p> <p><stop_bits> = 1* or 2 - the number of stop bits per character. The normal selection for 300 baud and higher is 1.</p> <p><flow_control> = H*, S, or N - the flow control method: Hardware (CTS/RTS), Software (X-ON, X-OFF) or None.</p> <p>With hardware flow control, RTS is turned on when the port is ready to transmit. Then, transmission begins when CTS becomes active. RTS remains on until <delay_value> expires after the last character is sent.</p> <p>With software or no flow control, RTS is not turned on, and transmission begins immediately.</p> <p><physical_interface> = 232*, 422, or 485 - the physical connection protocol for the port: RS-232, RS-422, or RS-485. RS-422 is equivalent to RS-485. All Series 90-30 PCMs support RS-232 only on COM1. IC693PCM300 supports RS-422/485 only on COM2.</p> <p><duplex_mode> = 2, 4*, or p - the type of physical connection: 2 = half duplex (2 wire for RS-422/485), 4 = full duplex (4 wire for RS-422/485), p = point-to-point. Available in PCM firmware version 3.00 or later.</p> <p>In point-to-point mode:</p> <ul style="list-style-type: none"> • The receiver for the specified port is always enabled. • When <physical_interface> = 422 or 485, all RS-485 line drivers for the specified port are enabled when the command is executed and remain on continuously. <p>* Default selection.</p> |

| Device | Initialization String |
|------------------------------------|--|
| COM1: COM2: (Continued) W | <p><duplex_mode> (Continued)</p> <p>In full duplex mode:</p> <ul style="list-style-type: none"> • The receiver for the specified port is always enabled. • When <physical_interface> = 422 or 485, the RS-485 line drivers for RTS and transmitted data outputs on the specified port are turned on immediately before transmitting and remain on until <delay_value> expires after the last character is sent. At all other times, these drivers are in their high-impedance state (tri-stated). <p>In half duplex mode:</p> <ul style="list-style-type: none"> • The receiver for the specified port is disabled immediately before transmitting and remains off until <delay_value> expires after the last character is sent. • When <physical_interface> = 422 or 485, the RS-485 line drivers for RTS and transmitted data outputs on the specified port are turned on immediately before transmitting and remain on until <delay_value> expires after the last character is sent. At all other times, these drivers are in their high-impedance state (tri-stated). <p><delay_value> = the time in milliseconds between the end of the last outgoing character and the time RTS is turned off (if applicable), RS-485 line drivers are tri-stated (if applicable), the receiver is enabled in half duplex mode (if applicable), and WAIT mode output statements complete execution. Default = 0. Available in PCM firmware version 3.00 or later.</p> <p><typeahead_size> = the typeahead buffer size in characters for the port. The port can accept up to one less than this number of characters without overflow before an application reads the port. When overflow occurs, any additional characters will be lost. Any size in the range 64 – 32750 bytes may be specified, but the maximum may be limited by available system memory. Default = 320. Available in PCM firmware version 3.00 or later.</p> |
| CPU:#5 | <p><PLC_access_password>,<disable_clock_sync> where:</p> <p><PLC_access_password> = the PLC access password for privilege level 2 or higher. If passwords are enabled in the PLC CPU and the PLC has passwords at level 2 and higher, the PCM will be unable to read or write PLC memory until the PCM sends a valid password. Passwords are case sensitive, and valid passwords may have upper case letters, numbers, and underbar ('_') characters only. If an empty string is specified for <PLC_access_password>, a password consisting of eight NUL characters will be sent to the PLC CPU. There is no default.</p> <p><disable_clock_sync> = N - disables backplane messages the PCM normally sends once per second to synchronize its internal time of day with the PLC CPU. Any character other than 'N' or 'n' enables clock synchronization. Available in PCM firmware version 4.03 or later.</p> <p>Some applications may be sensitive to the impact that clock synchronization messages have on PLC sweep time or backplane message rates. If these issues are more important than time of day accuracy, use this option. Default = synchronization enabled.</p> |

Examples:

```

I COM1: 9600,,,,S
I COM2: 38400,O,8,1,S,485,2,10,1024

```

The first example sets the port 1 data rate to 9,600 baud and selects software flow control. Selections for parity, data bits, and stop bits are the omitted items between the four consecutive commas; they are unchanged.

The second example sets port 2 for RS-485 two wire half duplex operation at 38,400 baud, odd parity, 8 data bits, and one stop bit; using software flow control, a ten millisecond time delay, and a 1024 character typeahead buffer.

```
I CPU:#5 PASSWD
I CPU:#5 MYPASWD,N
I CPU:#5 ,Y
```

The third example sets the PCM privilege to the access level protected by the password "PASSWD".

The fourth example sets the PCM privilege to the access level protected by "MYPASWD" and also disables PCM clock synchronization.

The last example re-enables PCM clock synchronization but has no effect on PLC access level.

J (format EEROM device)

Format: J ROM:

This command causes an electrically erasable ROM (EEROM) device installed in a PCM 301 to be erased and formatted as the file device ROM:. Once the EEROM has been formatted, files can be loaded to and run from it just as they are in RAM:. Note that only the PCM 301, IC693PCM301, supports an optional EEROM device.

Attempting to format an invalid device produces this message:

Unknown device

K (Kill a task)

Format: K <task_id>

This command stops the specified task and frees the resources it was using. The task is unlinked from all associated modules (all link counts are decremented). Timers used by the task are cancelled, pending ASTs are discarded, pending I/O is aborted, open files are closed, and memory used by the task is returned to the operating system.

The task ID argument must be in the range of 4 – 0F hexadecimal. If it is not, or if the task is not active, the PCM responds with the error message:

Can't terminate task

Note

PCM firmware version 2.51 or earlier limits the task number to a range of 4 – 7.

L (Load)

Format: L [**<options>**] **<pc_filename>** [**<pcm_filename>**]

This command directs the PCM to load the file specified by **<pc_filename>** to **<pcm_filename>** in the PCM RAM Disk. The file names are not case sensitive. If the optional **<pcm_filename>** is omitted, the PC file name will be used without any device or file path prefix. If the file already exists in the PCM, it will be overwritten.

If **<pc_filename>** has the extension **.EXE**, and the file begins with a valid MS-DOS relocatable EXE file header, the PCM will attempt to convert the file to an absolute load image in RAM.

Two optional qualifiers may be specified with the load command: display mode and protection mode. The display mode determines whether or not the file will be included in a module directory listing. The two modes are **normal (N)** and **hidden (H)**. Normal mode is the default.

The protection mode is used to determine whether the PCM file will be **volatile (V)**, **semi-volatile (S)**, or **protected (P)**. For certain file types (for example, **EXE** files), the protection level is fixed at P by default; a protection option in the load command is ignored. Data files are volatile by default, but this may be overridden by the load command.

If options are used in the load command, they follow the command directly, with no intervening spaces. For example, **LH** is used to load a hidden module, and **LHP** is used to load a protected hidden module.

Possible errors are:

```
File not found
Illegal module type
Insufficient memory
```

The PC file name may begin with a device name. If there is no device name, the default device PC: is assumed. A PC disk drive and file path specification may be included in **<pc_filename>**. However, if a PC disk drive is specified, the PC: device must also be explicitly specified; for example:

```
L PC:A:MYFILE.DAT
L PC:C:\MYDIR\MYFILE.EXE
```

M (create a memory Module)

Format: M <module_name> <size>

The create memory module command creates a PCM data module using the specified <module_name> and <size> arguments. The size is interpreted as a hexadecimal number.

The command has no effect if the module already exists. If the module does not exist, it will be created and initialized to all zeros. No checksum protection will be applied to the module; it may be freely read and written. The location of the module may change after a reset, but its contents will remain the same.

O (get LED configuration)

Format: O

This command is used to return the LED configuration for the PCM. Two words of binary data are returned. The first word holds the configuration of LED 1, and the second holds the LED 2 configuration. However, TERMF does not display the binary values correctly.

Note

The O command is provided mainly for PCOP and does not return meaningful data when invoked from interactive mode.

P (request status data)

Format: Px

This command requires a second uppercase letter (x), which specifies the type of status data requested. The information returned is explained in the following table:

| Option | Description |
|--------|---|
| PC | Show the status of the PCM configuration. If the last configuration completed without errors, the PCM returns the > prompt. If there were errors, the PCM returns an error string. |
| PD | Dump the operating status of PCM at the most recent PCM soft reset. When the PCM is soft reset, it saves the contents of its task control blocks plus the top 64 words of the stack for the task which was executing at the time and the top 32 words of all other tasks. The PD command causes the PCM to write this information to a binary file called PCMDUMP.OUT on the PC default directory. This data can be formatted as text with the PCMDUMP.EXE utility or the MegaBasic program PCMDUMP.PGM . The information is useful when the PCM resets or locks up unexpectedly. |
| PL | Show the PCM rack/slot location. The PCM returns two ASCII digits. The first digit specifies the rack number, and the second digit specifies the slot number. The ASCII digits may be followed by a string that contains the CPU ID. If the PLC CPU does not have an ID, no string is returned. If the PCM cannot establish communication with the CPU, the message "NO CPU" is sent. |
| PM | <p>Show the type and mode of the most recent PCM reset. The PCM returns two ASCII digits.</p> <p>The first digit specifies the type of the most recent PCM reset:</p> <ul style="list-style-type: none"> 0 = Powerup reset. 1 = Soft reset. 2 = Hard reset. 3 = ACFAIL reset. 4 = Reset caused by receipt of new soft switch data. 5 = Internal software error reset. 6 = Backplane (COMMREQ) reset. <p>The second digit specifies the type of configuration data used during the most recent reset:</p> <ul style="list-style-type: none"> 0 = User configuration data. 1 = Factory default configuration data. 2 = Logicmaster 90 configuration. 3 = A combination of Logicmaster 90 and factory default data. |
| PT | Show the status of PCM tasks. For each active task, the PCM returns the task number and the names of the task's code and environment modules. |

Q (set protection level)

Format: Q <file_name> <protection_level>

This command is used to change the protection level of a module or modules on the PCMRAM Disk.

| Level | Description |
|-------|---|
| 0 | Unprotect the file: the file is not checksum protected and may be freely written. The checksum is not verified on power up or reset. |
| 1 | Protect the file: a checksum is calculated for it, and it may no longer be written to. On power-up or reset, the checksum of the module is verified and, if it is not correct, the module is discarded. |

R (Run)

Format: **R** <file_name> [<options>]

This command causes the PCM to run the executable file specified by <file_name>. The following options are available:

| Option | Description |
|----------|---|
| >outchnl | Redirect standard output to the channel specified by outchnl . Choices are COM1:* , COM2: , RAM:<pcm_filename> , and PC:<pc_filename> , where <pcm_filename> and <pc_filename> are the names of PCM RAM Disk and PC files, respectively. A device must be explicitly specified for files; the colon is required. A <pc_filename> may contain a PC device and/or file path specification: R MYFILE.EXE >PC:C:[backsl]MYDIR[backsl]MYFILE.OUT |
| <inchnl | Take standard input from the channel specified by inchnl . Choices are identical to the ones for the >outchnl option. |
| ?errchnl | Redirect standard error output to the channel specified by errchnl . Choices are identical to the ones for the >outchnl option. |
| /Sxxxx | Use a stack size of xxxx hexadecimal bytes; 400 is the default. |
| /Dxxxx | Allocate xxxx hexadecimal bytes of data space to the task. The balance is reserved for RAM Disk or other tasks. |
| /Ex | Specify the executable task type; the default is 1 (priority based) . For PCM version 3.00 or later, a 2 (time slice) option is also available. |
| /Ix | Use a specified task ID value in the range 4 – 0F hexadecimal; by default the lowest unused value. Release 2.51 and earlier PCMs limit the task number to a range of 4 – 7. |
| /Txx | Allocate an execution time slice of xx hexadecimal milliseconds. Time slice options only apply to release 3.00 or later PCMs. |
| /Mname | Link the task to memory module name (may occur multiple times). |
| /B | Run the task in background mode. <u>Note:</u> If the task and the PCM command interpreter share a serial port, the command interpreter remains active and may interfere with the task. |
| /K | Keep the task's environment block in memory after task termination. |

* Default selection. The default selection is whatever port is being used by the command interpreter; normally, this is COM1:.

All strings in the command line which do not begin with the special characters (> , < , ? , or /) are assumed to be command line data strings. They are passed to the executable program for processing.

If the file specified by the R command is not present in the PCM RAM Disk, the PCM attempts to load it from the default device PC: using the specified file name. If it cannot find the file in either device, an error message is returned. Errors that can be returned are:

```
Module not found <file_name>
File not found
Insufficient memory
```

S (Save)

Format: **S** <pcm_filename> [<pc_filename>]

This command causes a file named <pcm_filename> in the PCM RAM Disk to be saved to a PC file. If the optional <pcm_filename> is omitted, the PCM file name will be used, and the file will be saved in the current directory of the current drive. If the file does not already exist on the PC, it is created; otherwise, the existing PC file is overwritten. The <pcm_filename> may include a PC disk drive and/or file path, as described above for the **L** (**Load**) command.

EXE files which have been loaded to PCM RAM no longer contain relocation information. They cannot be saved to PC files.

Possible errors are:

```
Module not found <file_name>
Cant save module <file_name>
```

U (reconfigure the PCM)

Format: **U** <configuration_file>

This command reconfigures the PCM according to the specified <configuration_file>. If any of the modules specified in the configuration file are missing, or if errors are encountered while initializing the new configuration, the PCM is placed in its factory default configuration.

The <configuration_file> argument must be either FDEF (factory) or UCDF (user). UDCF configuration data is stored only by PCOP. Any other file name results in one of these error messages:

```
Module not found <file_name>
Illegal module type <file_name>
```

V (Verify a file)

Format: **V** <file_name>

This command causes the PCM to verify the checksum of the specified file on the PCM RAM Disk. The file must have checksum protection. If the checksum calculated for the file matches the checksum stored in it, the PCM returns just the > prompt. If the checksums do not match, the PCM prints an invalid file error code, followed by the > prompt.

The **Q** command is used to enable and disable checksum protection for files.

W (Wait)

Format: W <time>

This command causes PCM command interpreter to wait for the specified time, in seconds, before initiating or responding to any activity on the serial ports. After the specified time, the PCM prints the > prompt or executes the next command in the **.BAT** file.

This command is provided to allow tasks running in background mode to access the PC file server. Since the PCM command interpreter and file server usually share the same serial port, the command interpreter has to be kept from using the port while the background task is using the file server. This is not a problem with tasks run in foreground mode, because the command interpreter is suspended until the foreground task completes.

This command is also useful for insuring that one task is completely initialized before starting another task. It is commonly used in **.BAT** files.

X (eXterminate file)

Format: X <file_name>

This command deletes a file named <file_name> in the PCM RAM Disk.

Possible errors are:

```
Module is use <file_name>
Module not found <file_name>
```

Note

The specified file is deleted immediately. There is no confirmation prompt, nor is there any method for recovering a deleted file.

Y (set upper memory limit)

Format: Y [<limit>]

PCM memory may be divided between the operating system and one or more applications. Setting a limit on the memory used by the operating system causes it to ignore all memory above the limit. It will never load programs or allocate memory buffers there. Application programs can determine the limit with the **GET_MEM_LIM** utility and access memory above the limit through the use of pointers or absolute addresses.

The optional <limit> argument is the amount of memory, in 16-byte paragraphs, to be retained by the operating system. It is specified as a hexadecimal value and must be at least 800 (32K bytes), to leave space for operating system data. If the specified value is too small or exceeds the total amount of memory on the PCM, this error message "Insufficient memory" is returned.

If the command is entered with no argument, the operating system uses all the RAM on the PCM, and the **GET_MEM_LIM** utility will indicate that no limit has been set. When a limit is set, it does not actually take effect until the PCM is reset.

Appendix D

PCM Batch Files

The PCM supports batch files which are superficially similar to those used with MS-DOS. You can specify a batch file to be executed interactively. In addition, you can create batch files which are executed automatically when power is applied or a soft reset occurs, or when a hard reset occurs. These files must be named **PCMEEXEC.BAT** and **HARDEXEC.BAT**, respectively.

Note

Your PCM must have firmware version 2.50 or greater in order to use PCM batch files.

PCM batch files may consist of any number of commands. All the commands described in appendix C, *PCM Commands*, may be used in batch files. Unlike MS-DOS, there are no commands, such as IF, ECHO, or GOTO, which work only in batch files.

Caution

When using PCM commands in batch files for PCMs with firmware release 2.51 or earlier, the command letter must be upper case. You can avoid batch file errors by using uppercase characters exclusively in PCM batch files.

Running Batch Files

Suppose you want to create a simple batch file that starts MegaBasic. Using a text editor on your computer, create a file called **TEST.BAT** which contains the single line:

```
R BASIC.EXE
```

After saving the file, invoke TERMF on your PC and then enter the PCM command interpreter interactive mode as described in appendix C, *PCM Commands*. Load the batch file to the PCM by typing:

```
L TEST.BAT
```

and pressing the Enter key.

Caution

Do not attempt to load batch files to the PCM using the MegaBasic **LOAD** command. The MegaBasic **LOAD** command converts files to MegaBasic program format, which is not understood by the PCM command interpreter. Using the MegaBasic **LOAD** command for batch files will prevent them from executing as expected.

To run the batch file, type **@TEST** and press the Enter key.

You should see the command in **TEST.BAT** followed by the MegaBasic start-up banner, indicating that the command interpreter executed the command to run MegaBasic. If you type **BYE** again, you should once again be communicating with the command interpreter in interactive mode. Verify this by pressing the Enter key, and note that the **>** prompt appears on a new line.

PCMEEXEC.BAT Files

When the PCM powers up or a soft reset occurs, the operating system looks for a file named **PCMEEXEC.BAT** in the RAM Disk. If it is not found, and the PCM has been configured by Logicmaster 90 software in BASIC or BAS/CCM mode, a new one is created. This file contains a single **R (Run)** command which instructs the command interpreter to allocate a specified block of PCM RAM to MegaBasic, to send a message to MegaBasic to execute a program named **BASIC.PGM**, and then to run the MegaBasic interpreter. If **BASIC.PGM** is found, it is then executed.

When the command interpreter starts, it executes the commands in **PCMEEXEC.BAT**, just as the MS-DOS command interpreter uses the **AUTOEXEC.BAT** file when MS-DOS is booted. However, **PCMEEXEC.BAT** is not executed following a hard reset.

HARDEXEC.BAT Files

When a hard reset occurs, the operating system looks for a file named **HARDEXEC.BAT** in the RAM Disk. If this file is not found, and the PCM has been configured for BASIC or BAS/CCM mode, a new **HARDEXEC.BAT** is created. It contains an **R (Run)** command which allocates a block of memory to MegaBasic and then starts the MegaBasic interpreter. In this case, no MegaBasic program name is specified, and MegaBasic starts in command mode.

User-Installed PCMEEXEC.BAT and HARDEXEC.BAT Files

You can create your own version of `PCMEEXEC.BAT` and/or `HARDEXEC.BAT`, and load them to the PCM RAM Disk. The commands you put in your `PCMEEXEC.BAT` will control the PCM whenever it powers up or a soft reset occurs. Similarly, commands in your `HARDEXEC.BAT` control the PCM when a hard reset occurs. Use your computer and any text editor which produces ASCII text files to create these files. Then use the **L (Load)** command, described in appendix C, *PCM Commands*, to load them to the PCM RAM Disk.

If it exists, `PCMEEXEC.BAT` is always run on power up and a soft reset, regardless of whether or not PCOP has stored User Configuration Data (UCDF) to the PCM. Although `PCMEEXEC.BAT` can be thought of as a configuration tool, its function is different from the UCDF. A UCDF can define the entire operating environment of the PCM, while `PCMEEXEC.BAT` is limited to defining the environment for application tasks.

Caution

Do not attempt to use both a `PCMEEXEC.BAT` file and UCDF configuration data. There are subtle interactions between them which can prevent the PCM from operating as expected.

The most common use of the `PCMEEXEC.BAT` file is to run application programs. Batch file commands can also be used to configure the user LEDs on the PCM and set the serial port communication parameters, as described in appendix C, *PCM Commands*.

Another way to create a `PCMEEXEC.BAT` file on the PCM RAM Disk is to use MegaBasic. This example creates a file to run `BASIC.EXE` with standard input, standard output, and standard error redirected to the NULL device. Type the statements below with no line numbers so that MegaBasic will execute them immediately.

```
Open #5, "PCMEEXEC.BAT"  
Print #5, "R BASIC.EXE TEST.PGM <NULL: <NULL: ?NULL: /D20000"  
Close #5
```


Appendix *E*

Example MegaBasic Program

The following program is a MegaBasic test program.

```
1000 Rem      MEGABASIC BACKPLANE INTERFACE TEST ROUTINES
1010 Rem      TEST.PGM

1040 Rem Define local variables to be used during backplane testing
1050 Def integer INTVBL1
1060 Def integer INTVBL2
1070 Def integer SUCCESS
1080 Def real REALVBL1
1090 Def real REALVBL2

1100 Rem Define "STATUS" Structure
1110 Struct integer CUR_STAT, integer STAT_HIST, integer STAT_TIME
1120 Dim STATSTR$(12)
1130 STABLE = 1
1140 READ_PENDING = 2
1150 READ_RECEIVED = 3
1160 READ_TIMEOUT = 4
1170 WRITE_PENDING = 5
1180 WRITE_RECEIVED = 6
1190 WRITE_TIMEOUT = 7
1200 NO_CPU = 11
1210 XFER_REJECT = 12
1220 SUCCESS = 0

1230 Rem First make sure that all parts of CPU memory can be accessed
1240 CHECKPLC "%R100", 12345
1250 CHECKPLC "%I001", 12345
1260 CHECKPLC "%RQ001", 12345
1270 CHECKPLC "%M001", 12345
1280 CHECKPLC "%T001", 12345
1290 CHECKPLC "%AI1", 12345
1300 CHECKPLC "%AQ1", 12345

1310 Rem Check to make sure that SYSTATUS$ function works correctly
1320 CHECKSTAT

1330 Rem Display Results of MegaBasic Test Program
1340 If SUCCESS = 0 then [
1350   Print "";Print "MegaBasic Test Program Completed Successfully"
1360 ]
1370 Else [
1380   Print "";Print "MegaBasic Test Program Failed:",SUCCESS," Failures Detected"
1390 ]
1400 End
```

```

1410 Rem Define The "CHECKPLC" Procedure
1420 Def proc CHECKPLC PLCLOC$, PATTERN%
1430 Local STATSTR$
1440 Errset XFERERR1
1450 SYSLINK INTVBL1, PLCLOC$
1460 Errset XFERERR2
1470 STATSTR$ = SYSTATUS$ (INTVBL1)
1480 If STATSTR$.CUR_STAT <> STABLE then Goto 1790
1490 SYSLINK INTVBL2, PLCLOC$
1500 Errset XFERERR3
1510 STATSTR$ = SYSTATUS$ (INTVBL2)
1520 If STATSTR$.CUR_STAT <> STABLE then Goto 1780
1530 INTVBL1 = PATTERN%
1540 INTVBL2 = 0
1550 SYSWRITE INTVBL1
1560 SYSREAD INTVBL2
1570 STATSTR$ = SYSTATUS$ (INTVBL1)
1580 If STATSTR$.CUR_STAT <> STABLE then Goto 1780
1590 STATSTR$ = SYSTATUS$ (INTVBL2)
1600 If STATSTR$.CUR_STAT <> STABLE then Goto 1780
1610 Errset XFERERR2
1620 UNLINK INTVBL2
1630 Errset XFERERR1
1640 UNLINK INTVBL1
1650 If INTVBL2 <> PATTERN% then [
1660   Print "CPU WORD ACCESS FAILURE FOR ", PLCLOC$
1670   SUCCESS += 1
1680 ]
1690 Else [
1700   Print "CPU WORD ACCESS SUCCESS FOR ", PLCLOC$
1710   Print "THE VALUE IN CPU WORD ADDRESS ", PLCLOC$, " IS ", INTVBL2
1720 ]
1730 Return
1740 XFERERR3: UNLINK INTVBL2
1750 XFERERR2: UNLINK INTVBL1
1760 XFERERR1: Print "BACKPLANE TRANSFER FAILURE FOR ", PLCLOC$; SUCCESS += 1
1770 Return
1780 UNLINK INTVBL2
1790 UNLINK INTVBL1
1800 Print "WRONG SYSTATUS VALUE FOR ", PLCLOC$; SUCCESS += 1
1810 Return
1820 Proc end

1830 Def proc CHECKSTAT
1840 Local STATSTR$
1850 Errset LINKERR2
1860 SYSLINK INTVBL1, "%R1"
1870 Errset
1880 STATSTR$ = SYSTATUS$(INTVBL1)
1890 If STATSTR$.CUR_STAT <> STABLE then [
1900   Print "Bad status for %R1 after syslink"; SUCCESS += 1
1910   UNLINK INTVBL1
1920   Return
1930 ]
1940 T1 = STATSTR$.STAT_TIME
1950 Wait (5)

```

```

1960 Errset WRITERR2
1970 SYSWRITE INTVBL1
1980 Errset
1990 STATSTR$ = SYSTATUS$(INTVBL1)
2000 If STATSTR$.CUR_STAT <> STABLE then [
2010   Print "Bad status for %R1 after syswrite"; SUCCESS += 1
2020   UNLINK INTVBL1
2030   Return
2040 ]
2050 T2 = STATSTR$.STAT_TIME
2060 Wait (5)
2070 If abs ( ( T2 - T1 ) - 5000 ) > 100 then [
2080   Print "Status time for syswrite is wrong"; SUCCESS += 1
2090 ]
2100 Else [
2110   Print "Good status time for syswrite"
2120 ]
2130 Errset READERR2
2140 SYSREAD INTVBL1
2150 Errset
2160 STATSTR$ = SYSTATUS$(INTVBL1)
2170 If STATSTR$.CUR_STAT <> STABLE then [
2180   Print "Bad status for ",PLCLOC$, " after sysread"; SUCCESS += 1
2190   Return
2200 ]
2210 T3 = STATSTR$.STAT_TIME
2220 If abs ( ( T3 - T2 ) - 5000 ) > 100 then [
2230   UNLINK INTVBL1
2240   Print "Status time for sysread is wrong"; SUCCESS += 1
2250 ]
2260 Else [
2270   Print "Good status time for sysread"
2280 ]
2290 Errset UNLINKERR2
2300 UNLINK INTVBL1
2310 Print "No status errors detected during transfer of %R1"
2320 Return
2330 LINKERR2:
2340 Print "Error detected for %R1 during syslink"; SUCCESS += 1
2350 Print errtyp, " ", errmsg$, " ", errpkg$
2360 Return
2370 WRITERR2:
2380 Print "Error detected for %R1 during syswrite"; SUCCESS += 1
2390 Print errtyp, " ", errmsg$, " ", errpkg$
2400 UNLINK INTVBL1
2410 Return
2420 READERR2:
2430 Print "Error detected for %R1 during sysread"; SUCCESS += 1
2440 Print errtyp, " ", errmsg$, " ", errpkg$
2450 UNLINK INTVBL1
2460 Return
2470 UNLINKERR2:
2480 Print "Error detected for %R1 during unlink"; SUCCESS += 1
2490 Print errtyp, " ", errmsg$, " ", errpkg$
2500 Return
2510 Proc end

```

Appendix F

TERMF File Descriptions

The following files are placed on the hard disk during the **TERMF INSTALL** procedure. The **AUTOEXEC.BAT** and **CONFIG.SYS** files are optional; you can select not to install them.

| File | Description |
|---------------------------|--|
| \PCOP | A directory. |
| CLEANUP.BAT | Cleans up (deletes) old files in the \PCOP directory. |
| DEFAULT.DAT | TERM settings for factory mode on a Workmaster computer. |
| INSTALL.DOC | Lists directories installed by TERMF INSTALL program. |
| TERM.DAT | TERM settings for factory mode on a Workmaster computer. |
| TERMF.EXE | Terminal emulator with file transfer protocol. |
| TERMSET.EXE | Installation utility for setting TERM parameters. |
| \PCOP\EXAMPLES.PCM | A directory. |
| ALM_RD.PGM | MegaBasic example reading %M bits. |
| BINARIES.DOC | Documents features of BYTESWAP.BIN and PORT_CTL.BIN . |
| BITFUNCS.ASM | Source file with documentation for BITFUNCS.BIN . |
| BITFUNCS.BIN | MegaBasic utilities package for bit string operations. |
| BYTESWAP.BIN | MegaBasic utilities for checksum and reversing byte order in words. |
| EXAMPLES.DOC | Documentation of MegaBasic .PGM packages. |
| GENERIC.DOC | Line number referenced documentation of GENERIC.PGM and GEN_TEST.PGM . |
| GENERIC.PGM | Definitions and procedures to access user references not directly supported by the PCM's backplane driver. |
| GEN_TEST.PGM | Sample program using GENERIC.PGM . |
| GRAPH.PGM | Sample graph program. |
| MBCRC.PGM | MegaBasic CRC checksum package. |
| PORT_CTL.BIN | PCM serial port control and status utilities. In firmware versions 2.51 and later, PORT_CTL.BIN is provided in firmware. This file is needed only with version 2.50 or earlier. |
| PRN_FLT.PGM | MegaBasic functions and procedures to analyze and print PLC and I/O fault records. |
| READ_FLT.PGM | MegaBasic functions and procedures to access PLC and I/O fault tables. |
| SAMPLE.PGM | Sample MegaBasic program. |
| TEST_FLT.PGM | How to use READ_FLT.PGM and PRN_FLT.PGM to read and display fault information. |
| UTILITY.DOC | Documentation for using UTILITY.PGM . |
| UTILITY.PGM | Procedures for gathering system information from the Series 90 CPU. |
| VT100.PGM | PCM MegaBasic extensions for VT100-style escape sequences. This file prints to STDOUT . |
| VT100_5.PGM | A companion file to VT100.PGM that prints to the device opening as #5. |
| \PCOP\UTILS | A directory. |
| ASMCHK.PGM | Refer to GFK-0256. |
| ASMDEFS.ASM | Refer to GFK-0256. |
| ASMPKG.BAT | Refer to GFK-0256. |
| CRUNCH.EXE | Compaction and encryption utility for MegaBasic program. |
| README.DOC | MegaBasic release notes. |

Appendix G

Synchronous Serial Mode Operation

This appendix outlines the technical information required to use the synchronous serial modes of the Series 90-70 PCMA3 and newer hardware. The PCMA3 has two separate ports (Port 1 and Port 2) which are isolated from each other and optically isolated from the host system. Only Port 1 is capable of synchronous mode serial operation.

The serial controller used on the PCMA3 is a NEC μ PD72001 Multiprotocol Serial Controller (MPSC). For detailed information on programming NEC72001, please contact NEC at 1-800-632-3531 (8 am to 4 pm Pacific time) and request a User's Manual for this product.

Port 1 Pin Assignments

Connector PL3 contains signals for both RS-232 and RS-485 communication circuits. The pin-out for RS-232 signals conforms to the RS-232 specification with the exception that pins normally unused for RS-232 are used for RS-485 signals.

Table G-1. Port 1 Pin Assignments: RS-232 (Synchronous Serial Mode Operation)

| RS-232 | | | | | |
|--------|---------------------|-------------|--------------------|-------------|--------|
| PIN | ASYNCHRONOUS | | SYNCHRONOUS | | I/O |
| | FUNCTION | SIGNAL NAME | FUNCTION | SIGNAL NAME | |
| 1 | Shield | - | Shield | - | - |
| 2 | TransmittedData | TD | TransmittedData | TD | Output |
| 3 | ReceivedData | RD | ReceivedData | RD | Input |
| 4 | Request to Send | RTS | Transmit Clock Out | TXCLKO | Output |
| 5 | Clear to Send | CTS | ReceiveClock | RXCLK | Input |
| 7 | SignalGround | 0V | Signal Ground | 0V | - |
| 8 | Data Carrier Detect | DCD | Transmit Clock In* | TXCLKI | Input |
| 20 | Data TerminalReady | DTR | Data TerminalReady | DTR | Output |

* Note: The Transmit Clock Input is used only with RS-232 synchronous mode communications. RS-422 synchronous mode uses only the RXCLK and TXCLKO lines.

Table G-2. Port 1 Pin Assignments: RS-422 (Synchronous Serial Mode Operation)

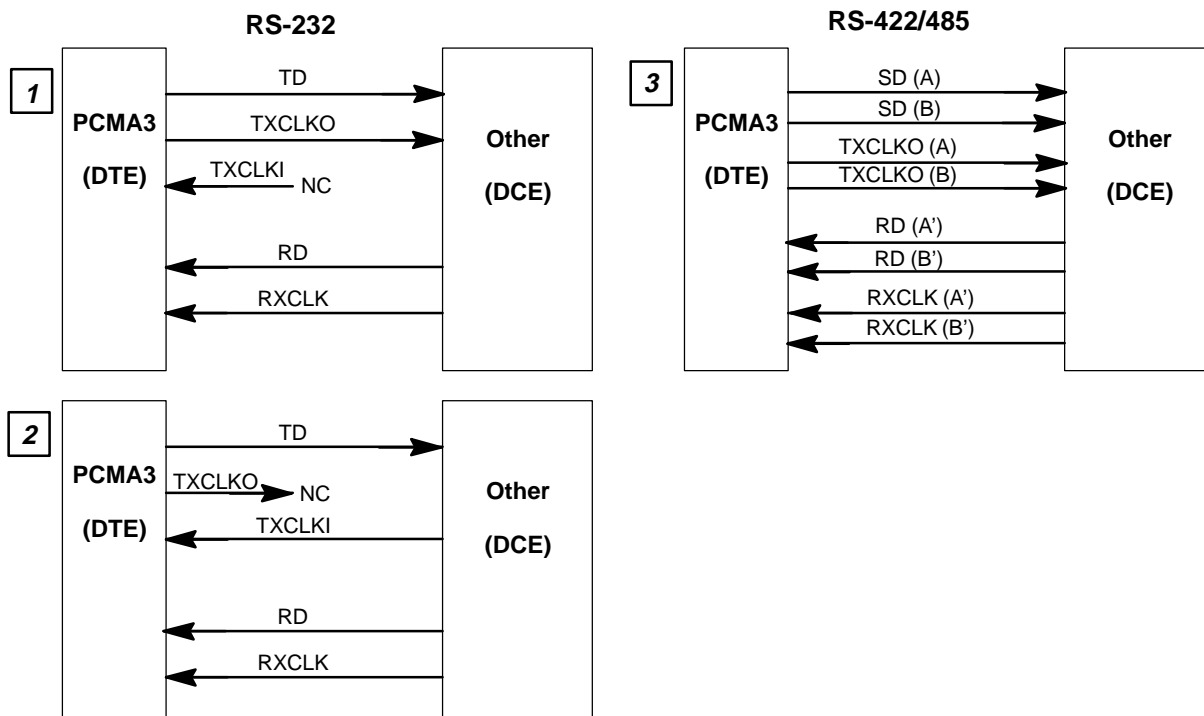
| RS-422 | | | | | |
|--------|---------------------|-------------|------------------------|-------------|--------|
| PIN | ASYNCHRONOUS | | SYNCHRONOUS | | I/O |
| | FUNCTION | SIGNAL NAME | FUNCTION | SIGNAL NAME | |
| 9 | Send Data (A) | SD (A) | Send Data (A) | SD (A) | Output |
| 10 | Request to Send (A) | RTS (A) | Transmit Clock Out (A) | TXCLKO (A) | Output |
| 11 | Clear to Send (A) | CTS (A') | Receive Clock (A) | RXCLK (A') | Input |
| 12 | Termination | - | Termination | - | - |
| 13 | Receive Data (A) | RD (A') | Receive Data (A) | RD (A') | Input |
| 21 | Send Data (B) | SD (B) | Send Data (B) | SD (B) | Output |
| 22 | Request to Send (B) | RTS (B) | Transmit Clock Out (B) | TXCLKO (B) | Output |
| 23 | Clear to Send (B) | CTS (B') | Receive Clock (B) | RXCLK (B') | Input |
| 24 | Termination | - | Termination | - | - |
| 25 | Receive Data (B) | RD (B') | Receive Data (B) | RD (B') | Input |

Synchronous Operation Modes for Port 1

Due to connector pin limitations, the asynchronous mode CTS/RTS lines are also used for the synchronous mode clock lines.

There are three possible synchronous operation modes for Port 1 of the PCMA3:

1. RS-232, Transmit Timing from PCM (TLCKO), Receive Timing from Other (RXCLK)
2. RS-232, Transmit Timing from Other (TLCLK1), Receive Timing from Other (RXCLK)
3. RS-422/485, Transmit Timing from PCM (TXCLKO), Receive Timing from Other (RXCLK)



Synchronous Mode PCMA3 (Port 1) Control Registers

The PCMA3 has several hardware registers for controlling the Synchronous mode operation. The I/O Addresses listed in the table below assume a programmable base address of 0000H.

Table G-3. Port 1 Control Registers

| I/O Address | DATA BITS | | | | | | | FUNCTION | Reset Value | |
|-------------|-----------|---|---|---|---|---|---|----------|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | | | 0 |
| 0106H | X | X | X | X | X | X | X | 232SL1 | 0 = RS-232 for Port 1, 1 = RS-422/485 for Port 1 | 0 |
| 010AH | X | X | X | X | X | X | X | 422EN1 | 0 = Port 1 RS-422/485 Drivers Disabled, 1 = Drivers Enabled | 0 |
| 0304H | X | X | X | X | X | X | X | TXCLK_EN | TXCLKI input enabled on DCD line 0 = Off, 1 = On | 0 |
| 0306H | X | X | X | X | X | X | X | SYNCH_EN | TXCLKO output enabled on RTS line(s) 0 = Off, 1 = On | 0 |
| 030CH | X | X | X | X | X | X | X | PCMA3 | Set PCMA3 = 1 then read Status register at 0101H, bit 3. If 0101H, bit 3 is also set, identifies PCMA3 as hardware platform | 0 |

NEC72001 I/O Addresses

The serial controller used on the PCMA3 is an NEC72001. The serial controller is located in I/O address space at addresses 0000H - 0006H (even addresses only). The table below shows the port assignments.

Table G-4. Serial Controller Port Assignments

| I/O Address | Function |
|-------------|--------------------------------|
| 0000H | Serial Port 1 Data |
| 0002H | Serial Port 2 Data |
| 0004H | Serial Port 1 Control / Status |
| 0006H | Serial Port 2 Control / Status |

NEC72001 Synchronous Clock Source Selection (CR15)

In synchronous mode, the PCMA3 hardware defines the $\overline{\text{STRxC}}$ input as the Receive Clock (RxCLK) and the $\overline{\text{TRxC}}$ pin as either Transmit Clock In (TXCLKI) or Transmit Clock Out (TXCLKO). Control Register 15 (CR15) in the NEC72001 is used to control the synchronous clock sources. The three operating modes listed previously correspond to the following settings of CR15 and the PCMA3 control registers defined in table G-3.

Table G-5. Synchronous Clock Source Selection

| Operation Mode | 232SL1 Bit In 0106H | 422EN1 Bit In 010AH | TXCLK_EN Bit In 0304H | SYNCH_EN Bit In 0306H | NEC72001:CR15 |
|---|---------------------------|-----------------------------------|-----------------------------|-----------------------------|---------------|
| 1. RS-232, Tx Clk Out, Rx Clk In | 0 | X | 0 | 1 | 15H |
| 2. RS-232, Tx Clk In, Rx Clk In | 0 | X | 1 | 1 | 08H |
| 3RS-422/485 Tx Clk Out, Rx Clk In | 1 | 1 = Drivers On 0 = Drivers Off | 0 | 1 | 15H |

For Further Information

For further information on the NEC72001 Serial controller, please refer to the NEC Users Manual for the μ PD72001-11 available from NEC Corporate Headquarters at 1-800-632-3531 8 am to 4 pm Pacific Time.

For technical support of the PCMA3, please call the GE Fanuc PLC Technical Support Hotline toll free at 1-800-GEFANUC.

Symbols

@ (execute a batch file) command, C-4

A

ACC_CODE\$\$, 5-16
 access codes for PLC memory types, 5-19
 ACCESS statement, 4-4, 4-11
 Access to PLC fault tables and PLC status, 5-21
 Accessing %P, %L, and password-protected data, 5-16
 Accessing PLC data, 4-15
 Adding a memory expansion board to the Series 90-70 PCM, 2-9
 Adding a PCM to the rack screen, 2-8
 Address allocations for the PCM, 5-102
 Addressing, 3-7
 data length, 3-8
 target/sourcememory addresses, 3-7
 Advanced MegaBasic programming, 5-1
 ALL_SENT_STATUS, 5-56
 ALM_RD.PGM, F-1
 Argument of wrong type (error code 102), 5-3
 Argument out of range (error code 101), 5-3
 ASMCHK.PGM, F-1
 ASMDEFS.ASM, F-1
 ASMPKG.BAT, F-1
 Asynchronous serial input and output, 5-91
 ATTR statement, 5-6, 5-12
 ATTR\$ function, 5-6, 5-11
 ATTRIB_OFF, 5-7
 Autoconfig, 1-13, 2-18
 AUTOEXEC.BAT, 6-9, D-2, F-1

B

B (configure LEDs) command, 4-25, C-4

Backing up your program, 4-5
 Backplane communication, 5-114
 backplane processing for the Series 90-30 PCM, 5-115
 backplane processing for the Series 90-70 PCM, 5-114
 Backplane interrupts, 5-60
 BKP_MSG, 5-60, 5-61
 BKP_XFER, 5-60, 5-61
 NOWAIT I/O statements, 5-94
 Backplane messages, 5-64
 data in MESSAGE_TXT\$, 5-83
 identifying the source of backplane messages, 5-82
 message type, 5-82
 source of the message, 5-82, 5-83
 structure of backplane messages, 5-65
 application defined, 5-65
 destination, 5-65
 message type, 5-65
 priority, 5-65
 reserved, 5-65
 source, 5-65
 status, 5-65
 to another PCM, 5-86
 value of MSG_TYPE%, 5-83
 Backplane processing for the Series 90-30 PCM, 5-115
 Backplane processing for the Series 90-70 PCM, 5-114
 Backplane timeout (error code 118), 5-4
 Backplane transfer failure, 6-4
 Backplane transfer failure (error code 112), 5-4
 Bad LED definition (error code 110), 5-3
 Bad timer definition (error code 108), 5-3
 BAS/CCM configuration mode, 2-10, 2-13, 4-3
 BASIC configuration mode, 2-10, 2-12, 4-3
 Basic extensions incompatible (error code 116), 5-4
 BASIC.PGM, 4-3, 4-5, 4-6, 4-26, 4-27, D-2
 Batch files, D-1
 creating a simple batch file, D-1
 HARDEXEC.BAT, D-1, D-2, D-3
 PCMEXEC.BAT, D-1, D-2, D-3
 running batch files, D-1
 TESTBAT, D-1
 Battery, 1-6

BINARIES.DOC, F-1
BITFUNCS.ASM, F-1
BITFUNCS.BIN, 5-50, F-1
BKP_MSG, 5-34, 5-60, 5-61, 5-76, 5-77
BKP_XFER, 5-60, 5-61
BLINK, 5-7
BOLD, 5-7
BREAK_OFF, 5-55
BREAK_ON, 5-55
BREAK_STATUS(), 5-55
Buffer space exceeded (error code 120), 5-4
Bus receiver module (BRM), 2-3
Bus transmitter module (BTM), 2-3
BYE command, 5-50
BYTESWAPBIN, 5-50, F-1

C

C (Clear the PCM) command, C-5
C_DN, 5-7
C_LE, 5-7
C_RT, 5-7
C_UP, 5-7
Cable and connector specifications, A-1
Cabling, A-5
 connecting the PCM to the programmer, 2-32
 RS-232 cables, A-7
 RS-422/RS-485 cables, A-10
Cabling information, PCM, A-1
Catalog numbers for expansion memory boards for the Series 90-70 PCM, 2-5
Catalog numbers for Series 90-30 PCM, 1-12
CCM COMMREQ data block, 3-19
CCM COMMREQ example, 3-25
CCM COMMREQ status word, 3-23
CCM comparisons, 3-9
CCM data Tx/Rx failure, 6-7
CCM ONLY configuration mode, 2-10, 2-11
CCM operation, 1-2, 3-1
 CCM commands not supported, 3-22
 CCM COMMREQ data block, 3-19
 CCM COMMREQ example, 3-25
 CCM COMMREQ status word, 3-23
 CCM comparisons, 3-9
 CCM data Tx/Rx failure, 6-7
 CCM status word, 3-18
 command block, 3-15
 data block, 3-17
 data block length, 3-16
 idle timeout value, 3-17
 maximum communication time, 3-17
 status pointer memory type, 3-17
 status pointer offset, 3-17
 wait/no wait flag, 3-16
 COMMREQ, 3-12
 COMMREQ IN parameter, 3-13
 COMMREQ OK and FT parameters, 3-14
 COMMREQ SYSID parameter, 3-13
 COMMREQ TASK parameter, 3-14
 communications request, 3-12
 data length, 3-8
 diagnostic status words, 3-6
 format of the COMMREQ function block, 3-13
 memory types not supported, 3-2
 NOWAIT mode, 3-12
 other COMMREQ faults, 3-15
 power-up delay, 3-15
 run to completion mode, 3-28
 scratch pad, 3-4
 Series 90 CCM memory addressing conventions, 3-7
 Series Five vs. Series 90 CCM memory types, 3-3
 Series One vs. Series 90 CCM memory types, 3-3
 service request (SVCREQ), 3-28
 SVCREQ example, 3-30
 SVCREQ FNC parameter, 3-28
 SVCREQ PARM parameter, 3-29
 system communications window, 3-28
 target memory types, 3-2
 target/sourcememory addresses, 3-7
 WAIT mode, 3-12
CCM status word, 3-18
CCM/PROG configuration mode, 2-10, 2-16
CHECK command, 4-8, 4-9

- CHECK_BREAK(), 5-55
- CHECK_CPU_HEALTH, 5-36, 5-46
- CHG_PRIV, 5-16, 5-17
- CHG_PRIV procedure, 5-83
- CLEANUPBAT, F-1
- CLEAR command, 2-10
- Clear the PCM (C) command, C-5
- CLEAR_TSS, 5-22, 5-28
- CLS statement, 5-6, 5-8
- CMRQ_DATAS, 5-76
- CMRQ_HDRS, 5-76
- CMRQ_RCVD%, 5-76, 5-77
- CMRQ_TXTS, 5-76
- Color graphics adapter (CGA), 2-26
- Command block, 3-15
 - data block, 3-17
 - data block length, 3-16
 - idle timeout value, 3-17
 - maximum communication time, 3-17
 - status pointer memory type, 3-17
 - status pointer offset, 3-17
 - wait/no wait flag, 3-16
- Command format for PCM commands, C-2
- Command interpreter, C-1
 - accessing the command interpreter, C-1
- COMMREQ, 3-12, 5-64
 - CCM COMMREQ example, 3-25
 - CCM COMMREQ status word, 3-23
 - CCM status word, 3-18
 - command block, 3-15
 - data block, 3-17
 - data block length, 3-16
 - idle timeout value, 3-17
 - maximum communication time, 3-17
 - status pointer memory type, 3-17
 - status pointer offset, 3-17
 - wait/no wait flag, 3-16
 - controlling COMMREQs, 5-76
 - data block, 3-19
 - format of the COMMREQ function block, 3-13, 5-68, B-1
 - IN parameter, 3-13, 5-68
 - interpreting COMMREQ messages, 5-67
 - MegaBasic blink LED program example, 5-75
 - MegaBasic COMMREQ command block, 5-71
 - MegaBasic COMMREQ example, 5-72
 - message types 130, 131, 194, and 195, 5-67
 - NOWAIT mode, 3-12, 5-68
 - OK and FT parameters, 3-14, 5-70
 - other COMMREQ faults, 3-15
 - power-up delay, 3-15
 - programming the PLC COMMREQ function block, 5-68
 - resetting the PCM from a PLC program, B-1
 - SYSID parameter, 3-13, 5-69
 - TASK parameter, 3-14, 5-69
 - WAIT mode, 3-12, 5-68
- Communication failure, 6-2
- Communication, unsolicited, 5-66
- Communications request, 5-64
 - CCM COMMREQ example, 3-25
 - CCM COMMREQ status word, 3-23
 - CCM status word, 3-18
 - command block, 3-15
 - data block, 3-17
 - data block length, 3-16
 - idle timeout value, 3-17
 - maximum communication time, 3-17
 - status pointer memory type, 3-17
 - status pointer offset, 3-17
 - wait/no wait flag, 3-16
 - controlling COMMREQs, 5-76
 - data block, 3-19
 - format of the COMMREQ function block, 3-13, 5-68, B-1
 - IN parameter, 3-13, 5-68
 - interpreting COMMREQ messages, 5-67
 - MegaBasic blink LED program example, 5-75
 - MegaBasic COMMREQ command block, 5-71
 - MegaBasic COMMREQ example, 5-72
 - message types 130, 131, 194, and 195, 5-67
 - NOWAIT mode, 3-12, 5-68
 - OK and FT parameters, 3-14, 5-70
 - other COMMREQ faults, 3-15
 - power-up delay, 3-15
 - programming the PLC COMMREQ function block, 5-68
 - SYSID parameter, 3-13, 5-69
 - TASK parameter, 3-14, 5-69
 - WAIT mode, 3-12, 5-68

- Communications request (COMMREQ), 3-12
 - Compacting programs, 4-12
 - Compatibility with MS-DOS MegaBasic, 4-7
 - Config mode, 2-9
 - CONFIG.SYS, 6-9, F-1
 - Configuration data, 2-9
 - Configuration file, local, 2-31
 - Configuration modes, 2-10
 - BAS/CCM, 2-10, 2-13
 - BAS/CCMmode, 4-3
 - BASIC, 2-10, 2-12
 - BASIC mode, 4-3
 - CCM ONLY, 2-10, 2-11
 - CCM/PROG, 2-10, 2-16
 - NONE, 2-10, 2-14
 - PCM CFG, 2-10, 2-14
 - PROG PRT, 2-10, 2-15
 - PROG/CCM, 2-10, 2-16
 - Configuration modes and the PCOP display, 2-17
 - Configuration problems, 6-8
 - configure LEDs (B) command, C-4
 - Configuring the PCM, 1-13
 - adding a memory expansion board to the Series 90-70 PCM, 2-9
 - adding a PCM to the rack screen, 2-8
 - autoconfig for the Series 90-30 PCM, 1-13
 - autoconfig of the Series 90-30 PCM, 2-18
 - config mode, 2-9
 - configuring the Series 90-30 PCM with the hand-held programmer, 2-19
 - default configuration for the Series 90-30 PCM, 1-13, 2-18
 - for CCM operation, 1-13
 - for MegaBasic operation, 1-14
 - I/O configuration rack screen, 2-6
 - local configuration file, 2-31
 - PCM configuration data, 2-9
 - PCM configuration modes, 2-10
 - PCM configuration modes and the PCOP display, 2-17
 - Series 90-30 PCM plug-and-go operation, 1-13, 2-18
 - TERMF installation and configuration, 2-24
 - using TERMSET to configure TERMF or PCOP, 2-26
 - with Logicmaster 90 software, 2-6
 - Connecting the PCM to the programmer, 2-32
 - Constants, integer and string constants, 5-6
 - CPS, 5-7
 - CPU name string too long (error code 115), 5-4
 - CPU_RESPONDING, 5-38
 - create a memory Module (M) command, C-11
 - CRUNCH, 4-12
 - CRUNCH.EXE, 4-12, 6-5, F-1
 - CTS_STATUS(), 5-56
 - CUR statement, 5-6, 5-10
 - CURS function, 5-6, 5-9
 - CUR_STAT codes, 4-22
 - NO_CPU, 4-22
 - READ_PENDING, 4-22
 - READ_RECEIVED, 4-22
 - READ_TIMEOUT, 4-22
 - STABLE, 4-22
 - WRITE_FINISHED, 4-22
 - WRITE_PENDING, 4-22
 - WRITE_RECEIVED, 4-22
 - WRITE_TIMEOUT, 4-22
 - XFER_REJECT, 4-22
 - CURHOMES, 5-7
 - Current status, 4-22
- ## D
- D (file Directory) command, 5-51, C-5
 - D-type connector, 1-12
 - Data block, 3-17, 3-19
 - Data block length, 3-16
 - Data coherency, 4-24
 - string record, 4-24
 - vector, 4-24
 - Data length, 3-8, 5-71
 - Data size, MegaBasic program and, 4-9
 - ACCESS, 4-11
 - changing the MegaBasic workspace size, 4-11

determining the size of a MegaBasic program, 4-10
DISMISS, 4-11
MegaBasic program packages, 4-11
SHOW, 4-10
STAT, 4-10
Data, accessing PLC, 4-15
DATE_OF_LAST_CPU_RESP, 5-38
Daughter board for the Series 90-70 PCM, 2-5
DCD_STATUS(), 5-56
Default configuration for the Series 90-30 PCM, 1-13, 2-18
DEFAULTDAT, 2-26, 2-33, 6-2, 6-9, F-1
Deleting PCM data files using TERMF, 5-50
Development package, PCM, 1-3
Device unavailable (error code 123), 5-4
Diagnosing serial communication problems, 2-33
Diagnostic status words, 3-6
DIR command, 4-8, 4-27
DIR_OFF, 5-56
DIR_ON, 5-56
DISMISS statement, 4-11
Dual port RAM, 5-109
Dual port RAM available for applications, 5-102
Duplex Mode
 Full Duplex, 5-53
 Half Duplex, 5-53
 Point-to-Point, 5-53
Duplicate remote variable (error code 111), 5-3
DW_DH_BOTS, 5-7
DW_DH_TOPS, 5-7
DW_SHS, 5-7

E

Encrypting programs, 4-12
END statement, 4-6
Enhanced graphics adapter (EGA), 2-26

ERASE_BOLS, 5-7
ERASE_BOTS, 5-7
ERASE_EOLS, 5-7
ERASE_LINES, 5-7
ERASE_SCREEN\$, 5-7
ERASE_TOPS, 5-7
Error codes, MegaBasic, 5-3
 argument of wrong type (102), 5-3
 argument out of range (101), 5-3
 Backplane timeout (118), 5-4
 backplane transfer failure (112), 5-4
 bad LED definition (110), 5-3
 bad timer definition (108), 5-3
 Basic extensions incompatible (116), 5-4
 buffer space exceeded (120), 5-4
 CPU name string too long (115), 5-4
 Device unavailable (123), 5-4
 duplicate remote variable (111), 5-3
 illegal backplane operation (117), 5-4
 illegal NOWAIT I/O operation (119), 5-4
 improper variable type (114), 5-4
 insufficient memory (103), 5-3
 invalid IOCTL string (122), 5-4
 missing argument (100), 5-3
 not enough timers (107), 5-3
 PCM hardware not present (106), 5-3
 remote variable unknown (104), 5-3
 serial port framing (126), 5-4
 serial port multiple (127), 5-4
 serial port overrun (125), 5-4
 serial port parity (124), 5-4
 task not active (109), 5-3
 too many remote variables (105), 5-3
 undefined local variable (113), 5-4
EXAM statement, 5-106, 5-109, 5-110
Example MegaBasic program, E-1
EXAMPLES.DOC, F-1
EXAMPLES.PCM, 4-27
Examples, MegaBasic programming, 4-26
execute a batch file (@) command, C-4
Exiting the MegaBasic interpreter, 4-6
Expansion memory, 2-5
Expansion memory boards for the Series 90-70 PCM, catalog numbers, 2-5
Expansion memory for the Series 90-70 PCM, 2-9
Extensions to MegaBasic, 1-2, 4-2, 5-1

eXterminate file (X) command, 5-51, C-16

F

F (show Free memory) command, C-5

Fault table entries, PLC, 6-3

Fault table header records

CLEAR_TSS\$, 5-28

FLTS_SINCE_CLEAR\$, 5-28

NUM_FLTSS\$, 5-28

NUM_READ\$, 5-28

Fault tables

access to PLC fault tables and PLC status, 5-21

fault table header records, 5-28

I/O fault address subrecords, 5-33

I/O fault table records, 5-29

I/O reference address subrecords, 5-32
known problems with fault table access, 5-33

PLC fault address subrecords, 5-32

PLC fault table records, 5-28

time stamp subrecords, 5-32

Features not supported by the PCM,
MegaBasic, 4-8

File descriptions, TERMF, F-1

ALM_RD.PGM, F-1

ASMCHK.PGM, F-1

ASMDEFS.ASM, F-1

ASMPKG.BAT, F-1

BINARIES.DOC, F-1

BITFUNCS.ASM, F-1

BITFUNCS.BIN, F-1

BYTESWAPBIN, F-1

CLEANUPBAT, F-1

CRUNCH.EXE, F-1

DEFAULTDAT, F-1

EXAMPLES.DOC, F-1

GEN_TESTPGM, F-1

GENERIC.DOC, F-1

GENERIC.PGM, F-1

GRAPH.PGM, F-1

INSTALL.DOC, F-1

MBCRC.PGM, F-1

PORT_CTL.BIN, F-1

PRN_FLTPGM, F-1

READ_FLTPGM, F-1

README.DOC, F-1

SAMPLE.PGM, F-1

TERM.DAT, F-1

TERMEXE, F-1

TERMSETEXE, F-1

TEST_FLTPGM, F-1

UTILITY.DOC, F-1

UTILITY.PGM, F-1

VT100.PGM, F-1

VT100_5.PGM, F-1

file Directory (D) command, 5-51, C-5

Files, loading and storing PCM data files
using TERMF, 5-50

FILL statement, 4-5, 5-109, 5-110

FLT_CHANGED%, 5-26

short status records, 5-30

FLT_CHANGED%(), 5-21

FLT_PRESENT%, 5-25

short status records, 5-30

FLT_PRESENT%(), 5-21

FLTS_SINCE_CLEAR\$, 5-28

FLUSH_PORT, 5-56

format EEROM device (J) command, C-9

Format of the COMMREQ function block,
3-13, 5-68, B-1

Full Duplex, 5-53

full duplex mode, C-8

Functional overview, 1-2

CCM operation, 1-2

extensions to MegaBasic, 1-2

hard reset, 1-3

MegaBasic operation, 1-2

PCM development package, 1-3

PCM operation modes, 1-3

PCM support utilities for personal
computers, 1-3

PCOP, 1-3

RAM disk, 1-2

restart/reset pushbutton, 1-3

soft reset, 1-3

TERMF, 1-3

TERMSET, 1-3

G

G (Get hardware ID) command, C-6

Gathering PLC information from
MegaBasic programs, 5-34

GEN_TESTPGM, 5-16, F-1

GENERIC.DOC, 5-16, F-1

GENERIC.PGM, 5-16, 5-64, 5-83, F-1
procedures

CHG_PRIV, 5-16, 5-17

- SMSG_WTEXT, 5-16, 5-18
- structures and constants
 - ACC_CODE\$, 5-16
 - R_PBMEM\$, 5-16
 - R_TMEM\$, 5-16
 - SMSG_TEXT\$, 5-16
 - W_PBMEM\$, 5-16
 - W_TMEM\$, 5-16
- Get hardware ID (G) command, C-6
- get LED configuration (O) command, C-11
- get PCM firmware revision number (H) command, C-6
- GET_MEM_LIM, C-17
- GRAPH.PGM, F-1

H

- H (get PCM firmware revision number) command, C-6
- Half Duplex, 5-53
- half duplex mode, C-8
- Hand-held programmer, 2-19
- Hard reset, 1-3
- HARDEXEC.BAT, 4-27, D-1, D-2, D-3
- Hardware
 - adding expansion memory to the Series 90-70 PCM, 2-5, 2-9
 - battery, 1-6
 - cable and connector specifications, A-1
 - cabling, A-5
 - catalog numbers for Series 90-30 PCM, 1-12
 - D-type connector, 1-12
 - installing a Series 90-30 PCM, 2-4
 - installing a Series 90-70 PCM, 2-4
 - installing other modules, 2-3
 - installing the PCM hardware, 2-3
 - LED indicators, 1-4
 - memory on the Series 90-30 PCM, 1-11
 - memory on the Series 90-70 PCM, 1-9
 - OK LED, 1-5
 - option connector, 1-10
 - overview of the Series 90-30 PCM, 1-11
 - overview of the Series 90-70 PCM, 1-9
 - PCM module description, 1-4
 - RS-232 cables, A-7
 - RS-422/RS-485 cables, A-10

- serial connectors, 1-7, A-2
- serial ports on the Series 90-30 PCM, 1-12
- serial ports on the Series 90-70 PCM, 1-10
- user-defined LEDs (USER1 and USER2), 1-6
- USER1 and USER2, 1-6
- watchdog timer, 1-5
- what you will need, 2-2
- WYE cable, 1-7, 1-12

I

- I (Initialize device) command, C-7
- I/O configuration rack screen, 2-6
- I/O fault address subrecords, 5-33
- I/O fault table records, 5-29
- I/O reference address subrecords, 5-32
 - structure
 - IOFA_BUS\$, 5-33
 - IOFA_BUS_ADD\$, 5-33
 - IOFA_PT_OFFSET\$, 5-33
 - IOFA_RACK\$, 5-33
 - IOFA_SLOT\$, 5-33
 - IORA_ADD\$, 5-32
 - IORA_SEG\$, 5-32
- Idle timeout value, 3-17, 5-71
- Illegal backplane operation (error code 117), 5-4
- Illegal NOWAITI/O operation (error code 119), 5-4
- Improper variable type (error code 114), 5-4
- IN_LENGTH, 5-56
- INCHRS function, 4-14, 6-6
- Initialize device (I) command, C-7
- INPUT statement, 4-14
- Input to the PCM serial ports, 4-14
- INPUT() function, 4-14
- INSTALL.DOC, F-1
- Installing the PCM, 2-1
 - adding expansion memory to the Series 90-70 PCM, 2-5, 2-9
 - cabling, A-5
 - configuring the PCM with Logicmaster 90 software, 2-6

- configuring the Series 90-30 PCM with the hand-held programmer, 2-19
 - connecting the PCM to the programmer, 2-32
 - diagnosing serial communication problems, 2-33
 - installing a Series 90-30 PCM, 2-4
 - installing a Series 90-70 PCM, 2-4
 - installing other modules, 2-3
 - installing the PCM hardware, 2-3
 - local configuration file, 2-31
 - TERMF installation and configuration, 2-24
 - using TERMSET to configure TERMF or PCOP, 2-26
 - what you will need, 2-2
- Insufficient memory (error code 103), 5-3
- Insufficient memory error, 6-4
- Integer constants, 5-6
 - ATTRIB_OFF, 5-7
 - BLINK, 5-7
 - BOLD, 5-7
 - C_DN, 5-7
 - C_LE, 5-7
 - C_RT, 5-7
 - C_UP, 5-7
 - REVERSE, 5-7
 - UNDERSCORE, 5-7
- Interactive mode, C-2
- Interfacing to the PCM hardware and Series 90 CPU, 4-13
- INTERRUPT function, 5-60
- Interrupts, 5-57
 - backplane interrupts, 5-60
 - BKP_MSG, 5-34
 - INTERRUPT statement, 5-57
 - PROCESS_READ, 5-94
 - PROCESS_WRITE, 5-94
 - timer interrupts, 5-59
 - TIMER1, 5-34, 5-39
 - TIMER2, 5-34, 5-41
 - TIMER3, 5-34, 5-43
 - TIMER4, 5-34, 5-46, 5-49
 - TIMER5, 5-34, 5-48
- Invalid IOCTL string (error code 122), 5-4
- IO_FAULT_HDR\$, 5-22
- IO_FAULT_HDR\$.NUM_READ\$, 5-23
- IO_FLT_ACT\$, 5-29
- IO_FLT_ADD\$, 5-29
- IO_FLT_CAT\$, 5-29
- IO_FLT_DESC\$, 5-29
- IO_FLT_GRP\$, 5-29
- IO_FLT_SPEC\$, 5-29
- IO_FLT_TBL, 5-22
- IO_FLT_TSS, 5-29
- IO_FLT_TYPES, 5-29
- IO_FT\$, 5-22, 5-29
- IO_REF_ADD\$, 5-29
- IOCTL, 5-52
 - parameters, 5-53
- IOCTL\$() function, 4-8
- IOFA_BUSS\$, 5-33
- IOFA_BUS_ADD\$, 5-33
- IOFA_PT_OFFSET\$, 5-33
- IOFA_RACK\$, 5-22, 5-33
- IOFA_SLOTS, 5-33
- IORA_ADD\$, 5-32
- IORA_SEG\$, 5-22, 5-32
- ## J
- J (format EEROM device) command, C-9
- ## K
- K (Kill a task) command, C-9
- Kill a task (K) command, C-9
- ## L
- L (Load) command, 5-51, C-10, D-3
- LED indicators, 1-4, 4-25
 - on Series 90-30 PCM, 1-5
 - on Series 90-70 PCM, 1-4
- LIST command, 4-9
- Load (L) command, 5-51, C-10, D-3
- LOAD command, 4-4, 4-9, 4-12, 4-27
- Loading MegaBasic programs, 4-4
- Loading PCM data files using TERMF, 5-50
- Local configuration file, 2-31

Local RAM, 1-9, 1-11
Logicmaster 90 software, 2-6
 adding a memory expansion board to the Series 90-70 PCM, 2-9
 adding a PCM to the rack screen, 2-8
 autoconfig for the Series 90-30 PCM, 1-13
 autoconfig of the Series 90-30 PCM, 2-18
 config mode, 2-9
 configuration data, 2-9
 configuration modes and the PCOP display, 2-17
 configuration problems, 6-8
 default configuration for the Series 90-30 PCM, 1-13, 2-18
 I/O configuration rack screen, 2-6
 PCM configuration modes, 2-10
 Series 90-30 PCM plug-and-go operation, 1-13, 2-18
Loss of characters, 6-6

M

M (create a memory Module) command, C-11
Maximum communication time, 3-17, 5-71
MBCRC.PGM, F-1
MDE (module delete) command, 2-10
MegaBasic, 4-1
 access to PLC fault tables and PLC status, 5-21
 accessing %P, %L, and password-protected data, 5-16
 accessing PLC data, 4-15
 accessing the PCM's LEDs, 4-25
 additions or extensions created by GE Fanuc, 5-1
 advanced programming, 5-1
 asynchronous serial input and output, 5-91
 backing up your program, 4-5
 backplane interrupts, 5-60
 BASIC.PGM, 4-3, 4-5, 4-6, 4-26, 4-27
 changing the MegaBasic workspace size, 4-11
 CHG_PRIV, 5-16, 5-17
 commands
 BYE, 5-50
 CHECK, 4-8, 4-9
 DIR, 4-8, 4-27
 LIST, 4-9

LOAD, 4-4, 4-9, 4-12, 4-27
RUN, 4-8
SAVE, 4-4, 4-9, 4-26
SHOW, 4-10
STAT, 4-10
COMMREQ command block, 5-71
 idle timeout value, 5-71
 length, 5-71
 maximum communication time, 5-71
 status pointer, 5-71
 wait/no wait flag, 5-71
compacting programs, 4-12
compatibility with MS-DOS MegaBasic, 4-7
controlling COMMREQs, 5-76
data coherency, 4-24
data types
 string record, 4-24
 vector, 4-24
determining the size of a MegaBasic program, 4-10
encrypting programs, 4-12
error codes, 5-3
 argument of wrong type (102), 5-3
 argument out of range (101), 5-3
 backplane timeout (118), 5-4
 backplane transfer failure (112), 5-4
 bad LED definition (110), 5-3
 bad timer definition (108), 5-3
 Basic extensions incompatible (116), 5-4
 buffer space exceeded (120), 5-4
 CPU name string too long (115), 5-4
 Device unavailable (123), 5-4
 duplicate remote variable (111), 5-3
 illegal backplane operation (117), 5-4
 illegal NOWAIT I/O operation (119), 5-4
 improper variable type (114), 5-4
 insufficient memory (103), 5-3
 invalid IOCTL string (122), 5-4
 missing argument (100), 5-3
 not enough timers (107), 5-3
 PCM hardware not present (106), 5-3
 remote variable unknown (104), 5-3
 serial port framing (126), 5-4
 serial port multiple (127), 5-4
 serial port overrun (125), 5-4
 serial port parity (124), 5-4
 task not active (109), 5-3
 too many remote variables (105), 5-3
 undefined local variable (113), 5-4
example program, E-1
exiting from MegaBasic, 5-50
exiting the MegaBasic interpreter, 4-6
extensions developed by GE Fanuc, 4-2
fault table header records, 5-28

- features not supported by the PCM, 4-8
- FLT_CHANGED%, 5-26
- FLT_PRESENT%, 5-25
- functions
 - ATTR\$, 5-6, 5-11
 - BREAK_STATUS(), 5-55
 - CHECK_BREAK(), 5-55
 - CTS_STATUS(), 5-56
 - CUR\$, 5-6, 5-9
 - DCD_STATUS(), 5-56
 - INCHR\$, 4-14, 6-6
 - INPUT(), 4-14
 - INTERRUPT, 5-60
 - IOCTL\$, 4-8
 - MV_CUR\$, 5-6, 5-13
 - NOWAIT READ, 6-6
 - NOWAIT_IO, 4-14
 - OUTPUT(), 4-14
 - SYSTATUS\$, 4-15, 4-20, 4-22, 4-23
- gathering PLC information from
 - MegaBasic programs, 5-34
- GEN_TESTPGM, 5-16
- GENERIC.DOC, 5-16
- GENERIC.PGM, 5-16
- getting started with the MegaBasic interpreter, 4-3
- HARDEXEC.BAT, 4-27
- I/O fault address subrecords, 5-33
- I/O fault table records, 5-29
- I/O reference address subrecords, 5-32
- input and output to the PCM serial ports, 4-14
- interfacing to the PCM hardware and Series 90 CPU, 4-13
- INTERRUPT statement, 5-57
- interrupts
 - BKP_MSG, 5-34
 - TIMER1, 5-34, 5-39
 - TIMER2, 5-34, 5-41
 - TIMER3, 5-34, 5-43
 - TIMER4, 5-34, 5-46, 5-49
 - TIMER5, 5-34, 5-48
- known problems with fault table access, 5-33
- loading and saving MegaBasic programs, 4-4
- loss of characters, 6-6
- MegaBasic application does not run, 6-2
- MegaBasic blink LED program example, 5-75
- MegaBasic COMMREQ example, 5-72
- MegaBasic program access to PCM dual port RAM, 5-109
- MegaBasic program and data size, 4-9
- MegaBasic Tx/Rx failure, 6-6
- modifying existing BASIC programs, 4-8
- MS-DOS version of MegaBasic, 4-2
- NOWAIT I/O statements, 5-91
 - example NOWAIT program, 5-98
 - NOWAIT_CLOSE, 5-91, 5-96
 - NOWAIT_OPEN, 5-91, 5-92
 - NOWAIT_READ, 5-91, 5-94, 5-96, 5-97
 - NOWAIT_READ_ABORT, 5-91, 5-97
 - NOWAIT_SEEK, 5-91, 5-96
 - NOWAIT_WRITE, 5-91, 5-94, 5-96, 5-97
 - NOWAIT_WRITE_ABORT, 5-91, 5-97
- optimizing backplane communication, 5-114
- PCMEEXEC.BAT, 4-11, 6-5
- PLC fault address subrecords, 5-32
- PLC fault table records, 5-28
- printing a MegaBasic text file, 4-9
- PRN_FLTPGM, 5-21, 5-22
- procedures
 - BREAK_OFF, 5-55
 - BREAK_ON, 5-55
 - CHECK_CPU_HEALTH, 5-36, 5-46
 - CHG_PRIV, 5-83
 - DIR_OFF, 5-56
 - DIR_ON, 5-56
 - READ_PLC_CPU_ID, 5-35, 5-45
 - READ_PLC_FAULT_BIT, 5-36, 5-48
 - READ_PLC_RUN_STATUS, 5-35, 5-43
 - READ_PLC_STATUS, 5-35, 5-39
 - READ_PLC_TIME_AND_DATE, 5-35, 5-41
 - RTS_OFF, 5-56
 - RTS_ON, 5-56
 - UTILITY_INIT, 5-49
- program packages, 4-11
- programming example using VME functions, 5-110
- programming examples, 4-26
- programming the PCM in MegaBasic, 4-2
- programming the PLC COMMREQ function block, 5-68
- RDEL_FAULT_TBL, 5-24
- READ_FAULT_TBL, 5-23
- READ_FLTPGM, 5-21, 5-22
- repeated transfers of a MegaBasic variable, 5-114
- SAMPLE.PGM, 4-26
- saving data through a power cycle or reset, 4-6
- screen formatting commands, 5-5
 - VT100.PGM, 5-5

- VT100.PGM functions and procedures, 5-6
- VT100.PGM integer and string constants, 5-7
- VT100_5.PGM, 5-5
- serial port control and status, 4-14
- short status records, 5-30
- SMSG_WTEXT, 5-16, 5-18
- standard or specified devices, 4-13
- statements
 - ACCESS, 4-4, 4-11
 - ATTR, 5-6, 5-12
 - CLS, 5-6, 5-8
 - CUR, 5-6, 5-10
 - DISMISS, 4-11
 - END, 4-6
 - EXAM, 5-106, 5-109, 5-110
 - FILL, 4-5, 5-109, 5-110
 - INPUT, 4-14
 - IOCTL, 5-52
 - MV_CUR, 5-6, 5-15
 - OPEN, 4-9
 - PRINT, 4-14
 - PROCESS_MESSAGE, 5-64, 5-82
 - PROCESS_READ, 5-94, 5-97
 - PROCESS_WRITE, 5-94, 5-97
 - PROCESS_XFER, 5-61
 - SET_LED, 4-25
 - STOP, 4-6
 - SYSLINK, 4-15, 4-17
 - SYSREAD, 4-15, 4-19, 4-20, 4-22, 4-23, 4-24
 - SYSWRITE, 4-15, 4-19, 4-20, 4-22, 4-23, 4-24
 - UNLINK, 4-15, 4-19, 4-24
- status record, 4-22
- string variables
 - CMRA_DATA\$, 5-76
 - CMRQ_HDR\$, 5-72, 5-76
 - CMRQ_RCVD%, 5-76, 5-77
 - CMRQ_TXT\$, 5-72, 5-76
- TEST_FLTPGM, 5-21, 5-22
- time stamp subrecords, 5-32
- timer interrupts, 5-59
- timers and logical interrupts, 5-57
- using a text editor to create MegaBasic programs, 4-9
- UTILITY.PGM file, 5-34
- variables, #SSTAT, 4-16
- VME functions, 5-100
 - VMERD, 5-100, 5-103
 - VMERMW, 5-100, 5-107
 - VMETS, 5-100, 5-108
 - VMEWRT, 5-100, 5-105
- VT100_5.PGM, 4-27
- WORD%, 5-27
- MegaBasic application does not run, 6-2
- MegaBasic operation, 1-2
- MegaBasic program packages, 4-11
- MegaBasic TX/RX failure, 6-6
- MegaBasicPasswords, 5-17
- MegaBasicService request processor, 5-18
- MegaBasicSet graphic rendition (SGR) control string, 5-11
- Memory
 - insufficient memory error, 6-4
 - on the Series 90-30 PCM, 1-11
 - on the Series 90-70 PCM, 1-9
 - request and access codes for PLC memory types, 5-19
- Memory addressing, 3-7
 - data length, 3-8
 - target/sourcememory addresses, 3-7
- Memory types, 3-2
 - memory types not supported, 3-2
 - request and access codes for PLC memory types, 5-19
 - Series Five vs. Series 90 CCM memory types, 3-3
 - Series One vs Series 90 CCM memory types, 3-3
- Missing argument (error code 100), 5-3
- Modes of operation, 1-3
- Modifying existing BASIC programs for MegaBasic, 4-8
- Module description
 - battery, 1-6
 - catalog numbers for Series 90-30 PCM, 1-12
 - D-type connector, 1-12
 - hardware overview of the Series 90-30 PCM, 1-11
 - hardware overview of the Series 90-70 PCM, 1-9
 - LED indicators, 1-4
 - OK LED, 1-5
 - option connector, 1-10
 - serial connectors, 1-7
 - serial ports on the Series 90-30 PCM, 1-12
 - serial ports on the Series 90-70 PCM, 1-10

- user-defined LEDs (USER1 and USER2), 1-6
- USER1 and USER2 LEDs, 1-6
- watchdog timer, 1-5
- WYE cable, 1-7, 1-12
- Module description, PCM, 1-4
- Monochrome display adapter (MDA), 2-26
- MS-DOS, compatibility with MS-DOS MegaBasic, 4-7
- MS-DOS version of MegaBasic, 4-2
- MV_CUR statement, 5-6, 5-15
- MV_CUR\$ function, 5-6, 5-13
- MYFILE.BAT, C-4

N

- NO_CPU, 4-22
- NONE configuration mode, 2-10, 2-14
- Not enough timers (error code 107), 5-3
- Notation conventions for PCM commands, C-3
- NOWAIT example program, 5-98
- NOWAIT I/O statements, 5-91
 - example NOWAIT program, 5-98
 - NOWAIT_CLOSE, 5-91, 5-96
 - NOWAIT_OPEN, 5-91, 5-92
 - NOWAIT_READ, 5-91, 5-94, 5-96, 5-97
 - NOWAIT_READ_ABORT, 5-91, 5-97
 - NOWAIT_SEEK, 5-91, 5-96
 - NOWAIT_WRITE, 5-91, 5-94, 5-96, 5-97
 - NOWAIT_WRITE_ABORT, 5-91, 5-97
- NOWAIT mode, 3-12, 5-68
- NOWAIT READ function, 6-6
- NOWAIT SYSREAD commands, example, 5-62
- NOWAIT_IO functions, 4-14
- NUM_FLT\$, 5-28
- NUM_READ\$, 5-28

O

- O (get LED configuration) command, C-11
- OK LED, 1-5

- OK LED not on, 6-1
- OPEN statement, 4-9
- Operation modes, 1-3
- Option connector, 1-10
- Output from the PCM serial ports, 4-14
- OUTPUT() function, 4-14

P

- P (request status data) command, C-12
- Password-protected data, 5-16
- Path, MS-DOS search, 2-25
- PCM batch files, D-1
 - creating a simple batch file, D-1
 - HARDEXEC.BAT, D-1, D-2, D-3
 - PCMEEXEC.BAT, D-1, D-2, D-3
 - running batch files, D-1
 - TESTBAT, D-1
- PCM CFG configuration mode, 2-10, 2-14
- PCM commands, C-1
 - @ (execute a batch file), C-4
 - accessing the command interpreter, C-1
 - B (configure LEDs), 4-25, C-4
 - C (Clear the PCM), C-5
 - command format, C-2
 - D (file Directory), 5-51, C-5
 - displaying a list of PCM commands, C-2
 - entering interactive mode, C-2
 - F (show Free memory), C-5
 - G (Get hardware ID), C-6
 - H (get PCM firmware revision number), C-6
 - I (Initialize device), C-7
 - J (format EEROM device), C-9
 - K (Kill a task), C-9
 - L (Load), 5-51, C-10
 - L (Load) command, D-3
 - M (create a memory Module), C-11
 - notation conventions, C-3
 - O (get LED configuration), C-11
 - P (request status data), C-12
 - Q (set protection level), C-13
 - R (Run), C-14
 - R (Run) command, D-2
 - R (run) command, 4-11
 - S (Save), 5-51, C-15
 - summary of PCM commands, C-3
 - U (reconfigure the PCM), C-15
 - V (Verify a file), C-15
 - W (Wait), C-16

- X (eXterminate file), 5-51
- X (eXterminate), C-16
- Y (set upper memory limit), C-17
- PCM development package, 1-3
- PCM hardware not present (error code 106), 5-3
- PCM module
 - battery, 1-6
 - cable and connector specifications, A-1
 - cabling, A-5
 - catalog numbers for Series 90-30 PCM, 1-12
 - D-type connector, 1-12
 - hardware overview of the Series 90-30 PCM, 1-11
 - hardware overview of the Series 90-70 PCM, 1-9
 - LED indicators, 1-4
 - memory on the Series 90-30 PCM, 1-11
 - memory on the Series 90-70 PCM, 1-9
 - OK LED, 1-5
 - option connector, 1-10
 - RS-232 cables, A-7
 - RS-422/RS-485 cables, A-10
 - serial connectors, 1-7, A-2
 - serial ports on the Series 90-30 PCM, 1-12
 - serial ports on the Series 90-70 PCM, 1-10
 - user-defined LEDs (USER1 and USER2), 1-6
 - USER1 and USER2 LEDs, 1-6
 - watchdog timer, 1-5
 - WYE cable, 1-7, 1-12
- PCM module description, 1-4
- PCM support utilities for personal computers, 1-3
- PCMEXEC.BAT, 4-11, 4-25, 6-5, D-1, D-2, D-3
- PCMEXTPGM, 5-39, 5-43, 5-45, 5-48
- PCOP, 1-3, 1-15
 - communication failure, 6-2, 6-3
 - configuration problems, 6-9
 - configuring and programming for MegaBasic, 4-3
 - DEFAULTDAT, 6-2, 6-9
 - PCOP does not go on-line, 6-2
 - troubleshooting, 6-2
 - using TERMSET to configure PCOP, 2-26
- PCOP does not go on-line, 6-2
- PCOP locks up, 6-9
- PCOP screen goes blank, 6-9
- PLC fault address subrecords, 5-32
 - structure
 - PLCFA_RACK\$, 5-32
 - PLCFA_SLOT\$, 5-32
 - PLCFA_UNIT\$, 5-32
- PLC fault table entries, 6-3
- PLC fault table records, 5-28
 - format
 - IO_FLT_ACT\$, 5-29
 - IO_FLT_ADD\$, 5-29
 - IO_FLT_CAT\$, 5-29
 - IO_FLT_DESC\$, 5-29
 - IO_FLT_GRP\$, 5-29
 - IO_FLT_SPEC\$, 5-29
 - IO_FLT_TS\$, 5-29
 - IO_FLT_TYPE\$, 5-29
 - IO_FT\$, 5-29
 - IO_REF_ADD\$, 5-29
 - PLC_FLT_ACT\$, 5-28
 - PLC_FLT_ADD\$, 5-28
 - PLC_FLT_ERROR_CODE\$, 5-28
 - PLC_FLT_GRP\$, 5-28
 - PLC_FLT_SPEC\$, 5-28
 - PLC_FLT_TS\$, 5-28
 - PLC_FT\$, 5-28
 - spare/reserved, 5-28, 5-30
 - SS_CONTROL_PROG_NUM\$, 5-30
 - SS_NUM_CONTROL_PROGS\$, 5-30
 - SS_PLC_STATUS\$, 5-30
 - SS_PRIV_LEVEL\$, 5-30
 - SS_PROGRAMMER_FLAGS\$, 5-30
 - SS_SWEEP_TIMES\$, 5-30
- PLC status, access to PLC fault tables and PLC status, 5-21
- PLC_CPU_ID, 5-38
- PLC_CPU_ID_VALID, 5-38
- PLC_DATE, 5-38
- PLC_FAULT_BIT, 5-38
- PLC_FAULT_BIT_VALID, 5-38
- PLC_FAULT_HDR\$, 5-22
- PLC_FAULT_HDR\$.NUM_READ\$, 5-23
- PLC_FLT_ACT\$, 5-28
- PLC_FLT_ADD\$, 5-28
- PLC_FLT_ERROR_CODE\$, 5-28
- PLC_FLT_GRP\$, 5-28
- PLC_FLT_SPEC\$, 5-28

- PLC_FLT_TBL, 5-22
- PLC_FLT_TSS, 5-28
- PLC_FT\$, 5-22, 5-28
- PLC_RUN_STATUS, 5-38
- PLC_RUN_STATUS_VALID, 5-38
- PLC_STATUS_WORD, 5-36
- PLC_STATUS_WORD_VALID, 5-38
- PLC_TIME, 5-38
- PLC_TIME_AND_DATE_VALID, 5-38
- PLCFA_RACK\$, 5-22, 5-32
- PLCFA_SLOTS, 5-32
- PLCFA_UNITS, 5-32
- Plug-and-go operation, 2-18
- point-to-point mode, C-7
- Poit-to-Point Mode, 5-53
- PORT_CTL.BIN, 5-50, 5-55, F-1
 - functions and procedures
 - ALL_SENT_STATUS, 5-56
 - FLUSH_PORT, 5-56
 - IN_LENGTH, 5-56
 - requires an ACCESS statement, 5-55
- Power-up delay, 3-15
- PRINT statement, 4-14
- PRINT_MSG, 5-87
- Printing a MegaBasic text file, 4-9
- PRN_FLTPGM, 5-21, 5-22, F-1
- PROCESS_MESSAGE, 5-17, 5-18
- PROCESS_MESSAGE statement, 5-64,
5-72, 5-76, 5-82
 - arguments, 5-64
- PROCESS_READ statement, 5-94, 5-97
- PROCESS_WRITE statement, 5-94, 5-97
- PROCESS_XFER, 5-61
 - arguments, 5-61
- PROG PRT configuration mode, 2-10, 2-15
- PROG/CCM configuration mode, 2-10,
2-16
- Program and data size, MegaBasic, 4-9
 - ACCESS, 4-11
 - changing the MegaBasic workspace
size, 4-11

- determining the size of a MegaBasic
program, 4-10
- DISMISS, 4-11
- MegaBasic program packages, 4-11
- SHOW, 4-10
- STAT, 4-10
- Program compaction utility, CRUNCH,
4-12
- Program packages, MegaBasic, 4-11
- Program, example MegaBasic, E-1
- Programming example using VME
functions, 5-110
- Programming examples, MegaBasic, 4-26
- Programming the PCM in MegaBasic, 4-2
- Programming the PLC COMMREQ
function block, 5-68
- Programming, advanced MegaBasic, 5-1
- PROM, 1-9, 1-11
- Pushbutton, restart/reset, 1-3

Q

- Q (set protection level) command, C-13

R

- R (Run) command, D-2
- R (run) command, 4-11, C-14
- R_PBMEM\$, 5-16
- R_TMEN\$, 5-16
- Rack screen, 2-6
- RAM, 1-9, 1-11
 - local RAM, 1-9
 - shared RAM, 1-9
- RAM disk, 1-2
- RAM, dual port, 5-109
- RDEL_FAULT_TBL, 5-21
- RDEL_FAULT_TBL, 5-24
 - known problems with fault table access,
5-33
- Read only memory, 3-4
- READ_FAULT_TBL, 5-21, 5-23
 - fault table header records, 5-28
 - CLEAR_TS\$, 5-28

FLTS_SINCE_CLEAR\$, 5-28
NUM_FLTS\$, 5-28
NUM_READ\$, 5-28
known problems with fault table access,
5-33
READ_FLTPGM, 5-21, 5-22, F-1
functions and procedures, 5-21
FLT_CHANGED%, 5-26
FLT_CHANGED%(), 5-21
FLT_PRESENT%, 5-25
FLT_PRESENT%(), 5-21
RDEL_FAULT_TBL, 5-21, 5-24
READ_FAULT_TBL, 5-21, 5-23
WORD%, 5-27
WORD%(), 5-21
shared definitions and constants, 5-22
CLEAR_TS\$, 5-22
IO_FAULT_HDR\$, 5-22
IO_FLT_TBL, 5-22
IO_FT\$, 5-22
IOFA_RACK\$, 5-22
IORA_SEG\$, 5-22
PLC_FAULT_HDR\$, 5-22
PLC_FLT_TBL, 5-22
PLC_FT\$, 5-22
PLCFA_RACK\$, 5-22
SHORT_STATUS\$, 5-22
SS_NUM_CONTROL_PROGS\$, 5-22
TS_SEC\$, 5-22
READ_PENDING, 4-22
READ_PLC_CPU_ID, 5-35, 5-45
READ_PLC_FAULT_BIT, 5-36, 5-48
READ_PLC_RUN_STATUS, 5-35, 5-43
READ_PLC_STATUS, 5-35, 5-39
READ_PLC_TIME_AND_DATE, 5-35, 5-41
READ_RECEIVED, 4-22
READ_TIMEOUT, 4-22
README.DOC, F-1
reconfigure the PCM (U) command, C-15
Remote variable unknown (error code
104), 5-3
Request codes for PLC memory types,
5-19
request status data (P) command, C-12
Reset blinks user LED1 or LED2, 6-1
Resetting the PCM from a PLC program,
B-1

Restart/reset pushbutton, 1-3
RESTORECUR\$, 5-7
REVERSE, 5-7
RS-232 cables, A-7
RS-422/RS-485 cables, A-10
RTS_OFF, 5-56
RTS_ON, 5-56
Run (R) command, C-14, D-2
RUN command, 4-8
Run mode errors, 6-6
Run to completion mode, 3-28

S

S (Save) command, 5-51, C-15
SAMPLE.PGM, 4-26, F-1
Save (S) command, 5-51, C-15
SAVE command, 4-4, 4-9, 4-26
SAVECUR\$, 5-7
Saving data through a power cycle or
reset, 4-6
Saving MegaBasic programs, 4-4
Scratch pad, 3-4
Screen formatting commands, 5-5
VT100.PGM, 5-5
VT100.PGM functions and procedures,
5-6
VT100.PGM integer and string
constants, 5-7
VT100_5.PGM, 5-5
SEND_MESSAGE, 5-86, 5-87
SEND_UNSOLICITED_MSG, 5-86, 5-87
Serial connectors, 1-7, A-2
for the Series 90-30 PCM, A-3
for the Series 90-70 PCM, A-2
Serial port control and status, 4-14
Serial port framing (error code 126), 5-4
Serial port multiple (error code 127), 5-4
Serial port overrun (error code 125), 5-4
Serial port parity (error code 124), 5-4
Serial port setup with IOCTL and
PORT_CTL.BIN, 5-52

- Serial port status functions
 - INPUT(), 4-14
 - OUTPUT(), 4-14
- Serial ports, 1-10, 1-12
 - four wire full duplex mode, C-7, C-8
 - input and output to the PCM serial ports, 4-14
 - point-to-point, C-7
 - serial port control and status, 4-14
 - two wire half duplex mode, C-7
- Serial ports,, two wire half duplex mode, C-8
- Series 90-30 PCM
 - autoconfig, 1-13, 2-18
 - catalog numbers, 1-12
 - configuration problems, autoconfig, 6-8
 - D-type connector, 1-12
 - default configuration, 1-13, 2-18
 - hardware overview, 1-11
 - memory, 1-11
 - plug-and-go operation, 1-13, 2-18
 - serial connector, A-3
 - serial ports, 1-12
 - WYE cable, 1-12
- Series 90-70 PCM
 - adding expansion memory, 2-5, 2-9
 - catalog numbers for expansion memory boards, 2-5
 - daughter board, 2-5
 - hardware overview, 1-9
 - memory, 1-9
 - option connector, 1-10
 - serial connectors, A-2
 - serial ports, 1-10
- Service request
 - example, 3-30
 - FNC parameter, 3-28
 - PARM parameter, 3-29
- Service request (SVCREQ), 3-28
- set protection level (Q) command, C-13
- set upper memory limit (Y) command, C-17
- SET_LED statement, 4-25
 - LED number, 4-25
 - operation code, 4-25
- SGR control string, 5-11
- Shared RAM, 1-9
- Short status records, 5-30
- SHORT_STATUS\$, 5-22
- SHOW command, 4-10
- show Free memory (F) command, C-5
- Showing PCM data files using TERMF, 5-50
- SMSG_TEXT\$, 5-16
- SMSG_WTEXT, 5-16, 5-18
- Soft reset, 1-3, 4-3
- Specified devices, 4-13
- SS_CONTROL_PROG_NUM\$, 5-30
- SS_NUM_CONTROL_PROG\$, 5-22, 5-30
- SS_PLC_STATUS\$, 5-30
- SS_PRIV_LEVEL\$, 5-30
- SS_PROGRAMMER_FLAG\$, 5-30
- SS_SWEEP_TIMES, 5-30
- STABLE, 4-22
- Standard devices, 4-13
- standard non-privileged access type, 5-101
- STAT command, 4-10
- STAT_HIST, 4-23
- STAT_TIME, 4-23
- Status history, 4-23
- Status pointer, 5-71
- Status pointer memory type, 3-17
- Status pointer offset, 3-17
- Status record, 4-22
 - current status, 4-22
 - status history, 4-23
 - status time, 4-23
- Status time, 4-23
- Status variable #SSTAT, 4-16
- Status words, diagnostic, 3-6
- STOP statement, 4-6
- Storing PCM data files using TERMF, 5-50
- String constants, 5-6
 - CP\$, 5-7
 - CURHOMES, 5-7
 - DW_DH_BOTS, 5-7
 - DW_DH_TOP\$, 5-7
 - DW_SH\$, 5-7
 - ERASE_BOL\$, 5-7
 - ERASE_BOTS, 5-7

- ERASE_EOL\$, 5-7
- ERASE_LINES\$, 5-7
- ERASE_SCREEN\$, 5-7
- ERASE_TOP\$, 5-7
- RESTORECUR\$, 5-7
- SAVECUR\$, 5-7
- SW_SH\$, 5-7
- String record, 4-24
- Support utilities, PCM for personal computers, 1-3
- SVCREQ, 3-28
 - example, 3-30
 - FNC parameter, 3-28
 - PARM parameter, 3-29
- SW_SH\$, 5-7
- SYSLINK statement, 4-15, 4-17, 5-72, 5-77
 - arguments
 - CPU symbol, 4-17
 - handle, 4-17
 - local name, 4-17
 - type, 4-17
 - examples, 4-18
 - using the SYSLINK statement, 4-18
- SYSREAD, short status records, 5-30
- SYSREAD statement, 4-15, 4-19, 4-20, 4-22, 4-23, 4-24, 5-60, 5-61, 5-64, 5-114, 5-115
 - arguments
 - frequency, 4-21
 - data coherency, 4-24
 - NOWAIT, 4-20
 - variable name, 4-20
- SYSTATUS\$ function, 4-15, 4-20, 4-22, 4-23, 5-60
- System communications window, 3-28
- System overview, 1-1
- System window, 3-28
- SYSWRITE statement, 4-15, 4-19, 4-20, 4-22, 4-23, 4-24, 5-60, 5-61, 5-64, 5-72, 5-76, 5-77, 5-114, 5-115
 - arguments
 - frequency, 4-21
 - data coherency, 4-24
 - NOWAIT, 4-20
 - variable name, 4-20
- T**
 - Target memory types, 3-2
 - memory types not supported, 3-2
 - Series Five vs. Series 90 CCM memory types, 3-3
 - Series One vs. Series 90 CCM memory types, 3-3
 - Target/sourcememory addresses, 3-7
 - Task not active (error code 109), 5-3
 - TERM.DAT, 2-26, 6-3, F-1
 - TERM, 1-3, 1-15, 4-26
 - access from the MS-DOS search path, 2-25
 - communication failure, 6-2, 6-3
 - DEFAULT.DAT, 6-2
 - Deleting data files, 5-50
 - descriptions of files placed on the hard disk during INSTALL, F-1
 - ALM_RD.PGM, F-1
 - ASMCHK.PGM, F-1
 - ASMDEFS.ASM, F-1
 - ASMPKG.BAT, F-1
 - BINARIES.DOC, F-1
 - BITFUNCS.ASM, F-1
 - BITFUNCS.BIN, F-1
 - BYTESWAP.BIN, F-1
 - CLEANUP.BAT, F-1
 - CRUNCH.EXE, F-1
 - DEFAULT.DAT, F-1
 - EXAMPLES.DOC, F-1
 - GEN_TEST.PGM, F-1
 - GENERIC.DOC, F-1
 - GENERIC.PGM, F-1
 - GRAPH.PGM, F-1
 - INSTALL.DOC, F-1
 - MBCRC.PGM, F-1
 - PORT_CTL.BIN, F-1
 - PRN_FLT.PGM, F-1
 - READ_FLT.PGM, F-1
 - README.DOC, F-1
 - SAMPLE.PGM, F-1
 - TERM.DAT, F-1
 - TERM.EXE, F-1
 - TERMSET.EXE, F-1
 - TEST_FLT.PGM, F-1
 - UTILITY.DOC, F-1
 - UTILITY.PGM, F-1
 - VT100.PGM, F-1
 - VT100_5.PGM, F-1
 - installation and configuration, 2-24
 - Loading data files, 5-50
 - local configuration file, 2-31

- Showing data files, 5-50
 - Storing data files, 5-50
 - troubleshooting, 6-2
 - using TERMSET to configure TERMF, 2-26
 - TERMFEEXE, F-1
 - TERMSET, 1-3, 2-26, 2-31, 6-9
 - TERMSETEXE, F-1
 - TESTBAT, D-1
 - TEST_FLTPGM, 5-21, 5-22, F-1
 - Text editor, using a text editor to create MegaBasic programs, 4-9
 - Time of day clock, disabling synchronization messages, C-8
 - Time stamp subrecords, 5-32
 - structure
 - TS_DAY\$, 5-32
 - TS_HOUR\$, 5-32
 - TS_MIN\$, 5-32
 - TS_MON\$, 5-32
 - TS_SEC\$, 5-32
 - TS_YEAR\$, 5-32
 - TIME_AND_DATE_OF_LAST_CPU_RESP_VALID, 5-38
 - TIME_OF_LAST_CPU_RESP, 5-38
 - Timer interrupts, 5-59
 - parameters, 5-59
 - TIMER1, 5-34, 5-39
 - TIMER2, 5-34, 5-41
 - TIMER3, 5-34, 5-43
 - TIMER4, 5-34, 5-46, 5-49
 - TIMER5, 5-34, 5-48
 - Timers, 5-57
 - Too many remote variables (error code 105), 5-3
 - Troubleshooting, 2-33, 6-1
 - backplane transfer failure, 6-4
 - CCM data Tx/Rx failure, 6-7
 - communication failure, 6-2
 - configuration problems, 6-8
 - insufficient memory error, 6-4
 - known problems with fault table access, 5-33
 - loss of characters/MegaBasicTx/Rx failure, 6-6
 - OK LED not on, 6-1
 - PCOP screen goes blank or PCOP locks up, 6-9
 - PLC fault table entries, 6-3
 - reset blinks user LED1 or LED2, 6-1
 - TS_DAY\$, 5-32
 - TS_HOUR\$, 5-32
 - TS_MIN\$, 5-32
 - TS_MON\$, 5-32
 - TS_SEC\$, 5-22, 5-32
 - TS_YEAR\$, 5-32
- ## U
- U (reconfigure the PCM) command, C-15
 - UCDF (user configuration data), 1-13, 2-10
 - Undefined local variable (error code 113), 5-4
 - UNDERScore, 5-7
 - UNLINK statement, 4-15, 4-19, 4-24
 - Unsolicited communication, 5-66
 - User configuration data (UCDF), 1-13, 2-10
 - User-defined LED indicators, 4-25
 - User-defined LEDs, USER1 and USER2, 1-6
 - User-installed PCMEEXEC.BAT and HARDEXEC.BAT files, D-3
 - USER_BKP_MSG_PROC, 5-72, 5-76, 5-77, 5-83, 5-84, 5-86, 5-87
 - USER1, 1-6, 4-25
 - USER2, 1-6, 4-25
 - Utilities, PCM support for personal computers, 1-3
 - UTILITY.DOC, F-1
 - UTILITY.PGM, 5-34, 5-39, 5-41, 5-43, 5-46, 5-48, 5-49, F-1
 - interrupts
 - BKP_MSG, 5-34
 - TIMER1, 5-34, 5-39
 - TIMER2, 5-34, 5-41
 - TIMER3, 5-34, 5-43
 - TIMER4, 5-34, 5-46, 5-49
 - TIMER5, 5-34, 5-48
 - procedures
 - CHECK_CPU_HEALTH, 5-36, 5-46
 - READ_PLC_CPU_ID, 5-35, 5-45

- READ_PLC_FAULT_BIT, 5-36, 5-48
 - READ_PLC_RUN_STATUS, 5-35, 5-43
 - READ_PLC_STATUS, 5-35, 5-39
 - READ_PLC_TIME_AND_DATE, 5-35, 5-41
 - UTILITY_INIT, 5-36, 5-49
 - shared constants and variables
 - CPU_RESPONDING, 5-38
 - DATE_OF_LAST_CPU_RESP, 5-38
 - PLC_CPU_ID, 5-38
 - PLC_CPU_ID_VALID, 5-38
 - PLC_DATE, 5-38
 - PLC_FAULT_BIT, 5-38
 - PLC_FAULT_BIT_VALID, 5-38
 - PLC_RUN_STATUS, 5-38
 - PLC_RUN_STATUS_VALID, 5-38
 - PLC_STATUS_WORD, 5-36
 - PLC_STATUS_WORD_VALID, 5-38
 - PLC_TIME, 5-38
 - PLC_TIME_AND_DATE_VALID, 5-38
 - TIME_AND_DATE_OF_LAST_CPU_R
ESP_VALID, 5-38
 - TIME_OF_LAST_CPU_RESP, 5-38
 - true/false, 5-36
 - UTILITY_INIT, 5-36, 5-49
- ## V
- V (Verify a file) command, C-15
 - Vector, 4-24
 - Verify a file (V) command, C-15
 - Video graphics array (VGA), 2-26
 - VME functions, 5-100
 - general VME information for the PCM, 5-101
 - programming example, 5-110
 - rules for VME bus operations in Series 90-70 PLCs, 5-101
 - VME function blocks for communicating with the PCM, 5-100
 - VMERD, 5-100, 5-103
 - example, 5-104
 - VMERMW, 5-100, 5-107
 - VMETS, 5-100, 5-108
 - VMEWRT, 5-100, 5-105
 - example, 5-106
 - VT100.PGM, 5-5, F-1
 - functions and procedures, 5-6
 - ATTR, 5-6, 5-12
 - ATTR\$, 5-6, 5-11
 - CLS, 5-6, 5-8
 - CUR, 5-6, 5-10
 - CUR\$, 5-6, 5-9
 - MV_CUR, 5-6, 5-15
 - MV_CUR\$, 5-6, 5-13
 - integer and string constants, 5-7
 - ATTRIB_OFF, 5-7
 - BLINK, 5-7
 - BOLD, 5-7
 - C_DN, 5-7
 - C_LF, 5-7
 - C_RT, 5-7
 - C_UP, 5-7
 - CP\$, 5-7
 - CURHOME\$, 5-7
 - DW_DH_BOT\$, 5-7
 - DW_DH_TOP\$, 5-7
 - DW_SH\$, 5-7
 - ERASE_BOL\$, 5-7
 - ERASE_BOT\$, 5-7
 - ERASE_EOL\$, 5-7
 - ERASE_LINES\$, 5-7
 - ERASE_SCREEN\$, 5-7
 - ERASE_TOP\$, 5-7
 - RESTORECUR\$, 5-7
 - REVERSE, 5-7
 - SAVECUR\$, 5-7
 - SW_SH\$, 5-7
 - UNDERSCORE, 5-7
 - VT100_5.PGM, 4-27, 5-5, F-1
- ## W
- W (Wait) command, C-16
 - W_PBMEM\$, 5-16
 - W_TMEM\$, 5-16
 - Wait (W) command, C-16
 - WAIT mode, 3-12, 5-68
 - Wait/no wait flag, 3-16, 5-71
 - Watchdog timer, 1-5
 - WORD%, 5-27
 - WORD%(), 5-21
 - Workspace size, MegaBasic, 4-11
 - WRITE_FINISHED, 4-22
 - WRITE_PENDING, 4-22
 - WRITE_RECEIVED, 4-22
 - WRITE_TIMEOUT, 4-22
 - WYE cable, 1-7, 1-12, A-3

X

X (eXterminate file) command, 5-51, C-16

XFER_REJECT, 4-22

Y

Y (set upper memory limit) command,
C-17