Programmable Control Products

Series One™ Programmable Controllers ASCII/BASIC Module

User's Manual

GFK-0249A *March, 1989*

GE Fanuc Automation

Programmable Control Products

Series One™ Programmable Controllers ASCII/BASIC Module

User's Manual

GFK-0249A *March, 1989*

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 and software, nor to provide for every possible contingency in connection with installation, operation, and 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 ASCII/BASIC Module for the Series One™ family of Programmable Logic Controllers is a compact, highly reliable single-slot module that can be used in either a Series One or Series One Plus Programmable Logic Controller (PLC). The ASCII/BASIC Module under control of the resident BASIC programming language, and with appropriate ladder logic, can exchange information with the PLC. This information can be the status of any Input or Output point, or internal reference which can be transferred to the ASCII/BASIC Module from the Series One or Series One Plus PLC. Additionally, the contents of a register can be transferred to the Series One Plus PLC from the ASCII/BASIC Module. With appropriate programming, the ASCII/BASIC Module can instruct the Series One PLC to read or change the status of any output point or any register.

Using this Manual

Chapters 1 through 5 describe the features of the ASCII/BASIC Module, including information on how to create, edit, and store programs. Chapters 6 through 12 provide a reference to the BASIC programming functions resident in the module. The BASIC language for the ASCII/BASIC Module is Intel®MCS® BASIC-52 with additional custom commands created by GE Fanuc - NA.

The organization of the manual, including chapter titles and content, is as follows:

Chapter 1. Introduction: Provides an overview of the ASCII/BASIC Module for the Series One™ family of PLC's. Included are a description of the module's features and functions.

Chapter 2. Installation and Configuration: This is a description of installation and configuration requirements for the ASCII/BASIC Module. Included are lists of programming functions, including the custom commands created by GE Fanuc - NA. The end of this chapter provides troubleshooting information in case you should experience any hardware problems.

Chapter 3. Creating and Storing BASIC Programs: This chapter provides information on recommended methods for creating and storing your BASIC programs. Included are hints for documenting, programming, and debugging BASIC programs.

Chapter 4. ABMHelper: This chapter provides the information required to use ABMHelper when developing your BASIC programs. ABMHelper is a program developed by GE Fanuc - NA to operate with the ASCII/BASIC Module. It provides terminal emulation capability, archival storage of BASIC programs using your computer's disk storage facilities, and provides a full screen editor to use when creating and editing your BASIC programs. It is designed to run under MS-DOS and an IBM compatible BASIC interpreter - either BASICA or GWBASIC. It should work satisfactorily with most IBM-PC and compatible computer systems. The ABMHelper program is available on a distribution diskette (catalog number IC641PBE504) and must be ordered as a separate line item.

Chapter 5. W-ED Editor: This chapter describes the use of the W-ED editor, which is the BASIC editor supplied with the ASCII/BASIC Module. W-ED is a full function text editor that can be used to develop BASIC programs for the ASCII/BASIC Module. It can also be used as a general text editor for other programming purposes.

Chapter 6. Commands: This chapter provides a description and examples of the commands available for BASIC programming, including the GE Fanuc - NA custom BASIC commands.

Chapter 7. Archival Storage Area File Commands: This chapter describes a group of commands that are used to access or store programs to either RAM or the Archival Storage Area (ASA) in the ASCII/BASIC Module.

Chapter 8. Statements: Descriptions of how to use Statements, and examples of using them in your BASIC program are provided in this chapter.

Chapter 9. Operators and Expressions: The use of Operators and Expressions to be used when programming mathematical operations into your BASIC programs are described in this chapter. An example of a line of code using each Operator or Command is included.

Chapter 10. String Operators: Descriptions and use of String Operators are provided in this chapter.

Chapter 11. Special Function Operators: A group of nine Special Function Operators is described in this chapter. These instructions provide access to various system parameters such as the real time clock and the frequency at which the system is operating.

Chapter 12. Error Messages: This chapter provides a list of error messages that may be generated during the running of your BASIC program. A definition of each error message and the reason that it was generated is included for information and reference.

Chapter 13. Application Examples: This chapter provides examples of creating BASIC programs and ladder logic for transferring data to and from the ASCII/BASIC Module.

Appendix A. Quick Reference Tables of Commands, Statements, and Operators: This is a handy reference to all of the BASIC instructions, including a brief description of their function.

Appendix B. Miscellaneous Programming Information: This Appendix contains information which is intended to clarify and help you understand various aspects of creating BASIC programs using MCS BASIC-52.

Appendix C. Other Software Packages: This Appendix describes the use of a commercially available terminal emulator package named VTERM, which can be used for transferring files between an ASCII/BASIC Module and a diskette on a computer.

Appendix D. ASCII Code List: This Appendix is a list of the standard ASCII codes.

Related Publications

- GEK-90842 Series One™ and Series One Plus User's Manual
- GEK-25373 Workmaster® Guide to Operation
- GFK-0075 Logicmaster™1 Family Programming and Documentation Software User's Manual
- GFK-0269 Series Five™ ASCII/BASIC Module User's Manual

NOTE

For additional information relating to BASIC-52, the INTEL 8052AH microcontroller, or assembly language programming, refer to the *MCS BASIC-52 Users Manual* (Order Number: 270010) from Intel Corporation.

Henry A. Konat
Senior Technical Writer

The Series One/Series One Plus PLC and associated modules have been tested and found to meet or exceed the requirements of FCC Rule, Part 15, Subpart J. The following note is required to be published by the FCC.

NOTE

This equipment generates, uses, and can radiate radio frequency energy and if not installed in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits of a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

The following note is required to be published by the Canadian Department of Communications.

NOTE

This digital apparatus does not exceed the Class A limits for radio noise emissions from digital apparatus set out in the radio interference regulations of the Canadian Department of Communications.

CHAPTER 1. INTRODUCTION

Function of the ASCII/BASIC Module 1-1
General Specifications 1-2
ASCII/BASIC Module Communications 1-2
ASCII/BASIC Module Configuration 1-3
BASIC Language 1-3
Modes of Operation for Memory Use 1-3
MCS BASIC-52 Reference Information 1-4

CHAPTER 2. INSTALLATION AND CONFIGURATION

ASCII/BASIC Module Hardware Features 2-1
Status Display 2-1
RS-232 Serial Communications Port 2-2
The RTS Signal 2-3
DIP Switch Configuration 2-5
Run/Trap Mode 2-6
Memory Usage For Program Development 2-6
Expanded Mode of Operation 2-6
Partitioned Mode of Operation 2-6
Use of Keywords 2-7
Commands Used Only in Partitioned Mode 2-8
GE Fanuc - NA Custom BASIC Commands 2-8
Sign On Message 2-8
Commands, Statements, and Operators Not Supported 2-9
Troubleshooting the ASCII/BASIC Module 2-9
Troubleshooting Guide 2-9
Replacing the Lithium Battery 2-10

CHAPTER 3. CREATING AND STORING BASIC PROGRAMS

Advantages of Using ABMHelper 3-1
SOURCE/NOSOURCE Configuration 3-1
NOSOURCE Mode of Operation 3-2
SOURCE Mode of Operation 3-2
SOURCE/NOSOURCE Sub Modes 3-3
"Quick" SOURCE Mode Editing 3-3
Configuring ABMHelper (Revision 1.02 and later) 3-4
Baud Rate Setup 3-4
Communications Port Setup 3-5
Time and Date Setup 3-5
Hints for Documenting BASIC Programs 3-5
General Programming/Debug Hints 3-6

CHAPTER 4. ABMHELPER

Features of ABMHelper 4-1
System Requirements 4-1
Contents of the ABMHelper Distribution Diskette 4-2
Creating a Backup Distribution Diskette 4-2
Creating a Separate W-ED Editor Diskette 4-2
Creating a Workable ABMHelper Diskette (Floppy Disk Based System) 4-3

CHAPTER 4. ABMHELPER (cont)
Installing W-ED and ABMHelper on a Hard Disk System 4-3
Using ABMHelper 4-4
Running From Floppy Disk 4-4
ABMHelper Start-Up 4-5
Menu Option 1 - Terminal Mode 4-6
Menu Option 2 - Save Program to Disk 4-9
Menu Option 3 - Load a Program from Disk 4-9
Menu Option 4 - Merge a Disk File Into Program 4-9
Menu Option 5 - Delete a Range of Lines 4-9
Menu Option 6 - Directory of Disk Files 4-9
Menu Option 7 - Temporary Exit to DOS 4-10
Menu Option 8 - Set/View Time and Date 4-10
Menu Option 9 - Finished With ABMHelper 4-10
Using User-Supplied Routines with ABMHelper 4-10
Modifying ABMHelper for use with Fast Computers 4-11
Troubleshooting ABMHelper 4-12

CHAPTER 5. THE W-ED EDITOR
Editor Command Summary 5-1
Changing W-ED's Configuration 5-1
A Quick Introduction to W-ED 5-2
Starting W-ED From DOS 5-2
W-ED Function Keys 5-3
Other W-ED Key Actions 5-3
W-ED Features 5-3
Command Line and Text Area 5-4
Block or Cut and Paste Operations 5-4
Tabbing and Margins 5-4
Extended ASCII (Graphics) Characters 5-4
Printer Paging Control 5-5
Text Manipulation 5-5
W-ED Messages 5-6

CHAPTER 6. COMMANDS
Command: RUN(cr) 6-1
Command: CONT(cr) 6-1
Command: LIST(cr) 6-2
Command: NEW(cr) 6-3
Command: NULL [integer](cr) 6-3
Description of GE Fanuc - NA Custom Commands 6-3
Transfer Command: TRANSFER R/W, FORMAT, ABMLOC, FLAG 6-4
Functions Used in RUN Mode 6-5
LSTR 6-6
SPLIT 6-6
MOVE 6-6
SWITCH 6-7
MATCH 6-7
INPB 6-8

GFK-0249

CHAPTER 6. COMMANDS (cont)
Commands Used in Command (Monitor) Mode 6-8
RENUM@ 6-8
DELPRM 6-9

CHAPTER 7. **ARCHIVAL STORAGE AREA FILE COMMANDS**
Commands: RAM(cr) and ROM (integer) (cr) 7-1
Command: XFER(cr) 7-1
Command: BURN(cr) 7-2

CHAPTER 8. **STATEMENTS**
Statement: CLEAR 8-1
Statements: CLEARI and CLEARS 8-1
Statements: CLOCK1 and CLOCK0 8-2
Statements: DATA - READ - RESTORE 8-3
Statement: DIM 8-4
Statements: DO - UNTIL [rel expr] 8-5
Statements: DO - WHILE [rel expr] 8-6
Statement: END 8-6
Statements: FOR - TO - {STEP} - NEXT 8-7
Statements: GOSUB [ln num] - RETURN 8-8
Statement: GOTO [ln num] 8-9
Statements: ON [expr] GOSUB [ln num],[ln num], . . . [ln num] ON [expr] GOTO [ln num],[ln num], . . . [ln num] 8-10
Statement: IF - THEN - ELSE 8-10
Statement: INPUT 8-11
Statement: LET 8-13
Statement: ONERR [ln num] 8-14
Statement: ONTIME [expr],[ln num] 8-14
Statement: PRINT or P. 8-16
Special Print Formatting Statements 8-16
Statements: PH0., PH1. 8-18
Statement: PUSH [expr] 8-18
Statement: POP [var] 8-19
Statement: REM 8-20
Statement: RETI 8-20
Statement: STOP 8-21
Statement: STRING [expr],[expr] 8-21
Statement: IDLE 8-22
Statement: RROM (integer) 8-22
Statements: ST@ [expr] and LD@ [expr] 8-24

CHAPTER 9. **OPERATORS AND EXPRESSIONS**
Dual Operand Operators 9-1
Single Operand Operators - General Purpose 9-3
Single Operand Operators - Log Functions 9-4
Single Operand Operators - Trig Functions 9-5
Precedence of Operators 9-6
How Relational Expressions Work 9-6

GFK-0249

CHAPTER 10. STRING OPERATORS
- ASC( ) 10-1
- CHR( ) 10-2

CHAPTER 11. SPECIAL FUNCTION OPERATORS
- CBY([expr]) 11-1
- DBY([expr]) 11-1
- XBY([expr]) 11-1
- GET 11-1
- TIME 11-2
- XTAL 11-3
- System Control Values 11-3

CHAPTER 12. ERROR MESSAGES
- BAD SYNTAX 12-1
- BAD ARGUMENT 12-1
- ARITH. UNDERFLOW 12-1
- ARITH. OVERFLOW 12-2
- DIVIDE BY ZERO 12-2
- NO DATA 12-2
- CAN'T CONTINUE 12-2
- PROGRAMMING 12-2
- A-STACK 12-2
- I-STACK 12-2
- ARRAY SIZE 12-3
- MEMORY ALLOCATION 12-3

CHAPTER 13. APPLICATION EXAMPLES
- Example 1 - Transferring Data Between the ASCII/BASIC Module and the Series One/One Plus CPU 13-1
  - Using the Transfer Command 13-1
    - Example of TRANSFER Command 13-1
    - Transferring Register Data 13-2
- Example 2 - Sending Commands to an OIT 13-7

APPENDIX A. Commands, Statements, and Operators: Reference Tables A-1

APPENDIX B. Miscellaneous Programming Information B-1
- Floating Point Format B-1
- Storage Allocation B-1
- Format of an MCS BASIC-52 Program B-3

APPENDIX C. Other Software Packages C-1
- Using a Software Package Other Than ABMHelper C-1
  - Transfer a Program from ASCII/BASIC Module Memory to Diskette C-2
  - Retrieve a Program on Diskette and Store to the ASCII/BASIC Module C-2

APPENDIX D. ASCII Code List D-1

Figure			
	1-1	Series One PLC ASCII/BASIC Module	1-1
	2-1	RS-232 Communications Port Configuration	2-2
	2-2	Location and Replacement of Lithium Battery	2-10

Table			
	2-1	Status Indicator Definitions	2-1
	2-2	Serial Port Pin Definitions	2-3
	2-3	DIP Switch Configuration and Definition	2-5
	2-4	BASIC Commands Supported in All Modes	2-7
	2-5	GE Fanuc - NA Custom BASIC Commands	2-8
	4-1	Examples of Systems Tested	4-1
	4-2	Interactive/Edit Mode Control Functions	4-8
	4-3	F1 - F10 Function Keys	4-8
	5-1	PC Keyboard Conversion Chart	5-1
	5-2	Default Settings for W-ED Editor	5-2
	5-3	Block Operations	5-4
	6-1	R/W Parameter Description	6-4
	6-2	I/O Reference Ranges	6-4
	13-1	TRANSFER Command Parameters	13-1
	A-1	Commands	A-1
	A-2	Statements	A-1
	A-3	Dual Operand Operators	A-2
	A-4	Single Operand Operators	A-2
	A-5	Special Function Operators	A-3
	A-6	Stored Constants	A-3
	A-7	GE Fanuc - NA Custom BASIC Commands	A-3
	B-1	Sample Floating Point Format	B-1
	B-2	Location and Definition of Pointers	B-1
	B-3	Storage Allocation	B-2
	B-4	Line Format	B-3
	D-1	ASCII Code List	D-1

Chapter 1 Introduction

GFK-0249

The ASCII/BASIC Module (catalog number IC610ABM100) for the Series One™ family of Programmable Logic Controllers (PLC) is a compact module requiring only a single slot in a Series One or Series One Plus PLC rack. The ASCII/BASIC Module can be used in either a Series One or Series One Plus PLC control system. However, the expanded instruction set (data operations), which is a standard feature of the Series One Plus PLC, allows more efficient use of all of the features of the module.

Function of the ASCII/BASIC Module

The ASCII/BASIC Module under control of the resident BASIC programming language, and with appropriate ladder logic, can exchange information with the PLC. This information can be the status of any Input or Output point or internal reference, which can be transferred to the ASCII/BASIC Module from the Series One or Series One Plus PLC. Additionally, the contents of a register can be transferred to the Series One Plus PLC from the ASCII/BASIC Module. With appropriate programming, the ASCII/BASIC Module can instruct the Series One PLC to read or change the status of any output point or any register.

a42677

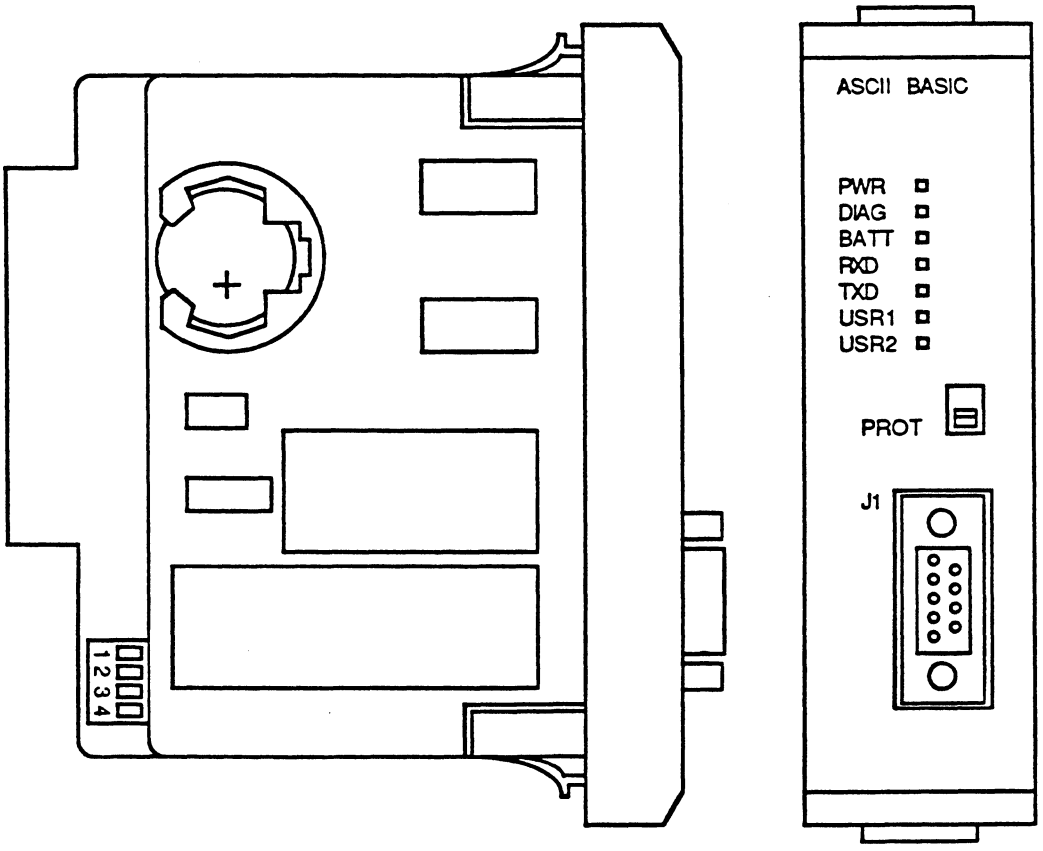


Figure 1-1. Series One PLC ASCII/BASIC Module

General Specifications

General specifications for the Series One ASCII/BASIC Module are as follows:

Operating Temperature	0°C to 60°C (32°F to 140°F)
Storage Temperature	-20°C to 70°C (-4°F to 158°F)
Humidity	5% to 95% (non-condensing)
Power requirements	300 mA, maximum (supplied by the 9 V dc output of the Series One power supply)
Environment	No corrosive gases
Meets Agency Standards For:	
Radio Frequency Interference	FCC Class A, part 15, subpart J
Shock	Mil-std 810C method 514.2, rank F JIS.C 0912 10 G
Vibration	Mil-std 810C method 516.2 JIS.C 0912 10 G
Noise Immunity	NEMA ICS3-304 RFI - 150, 450 Mhz Impulse 1000 V, 1 µs pulses
Clock Rate	11.0592 Mhz
Memory	56K RAM, Battery backed
Battery	Lithium (catalog number IC610BAT100), replaceable
Battery Life	One year, worst case. Three years, typical. Battery should be replaced within one week of the BATT indicator turning on. Once the battery has been removed from the module, it must be replaced within thirty minutes to ensure that RAM memory contents are maintained.
BASIC Language	Intel® MCS® BASIC 52 with additional commands by GE Fanuc - NA
Communications	Through a built-in RS-232 Serial Port (RTS included).
Baud Rates	300 to 19,200 baud, selectable through programming
Diagnostic/Status	LEDs controlled by CPU with additional LEDs controlled by user's BASIC program.

®Registered Trademark of Intel corporation

ASCII/BASIC Module Communications

The ASCII/BASIC Module has a built-in serial RS-232 communications port, which gives it the ability to interface to any ASCII device. These devices can be dumb CRT terminals, printers, modems, robots, other ASCII/BASIC Modules (including the Series Five™ ASCII/BASIC Module) and numerous other serial devices. With proper programming and ladder logic implementation, the ASCII/BASIC Module can communicate with Series One Data Communications Units, Series Three™ Data Communications modules, and Series Five and Series Six™ Communications Control Modules. ***Data sent to this port must be in ASCII format, unless using the INPB instruction.*** A table of the standard ASCII codes can be found in Appendix E in this manual.

ASCII/BASIC Module Configuration

An ASCII/BASIC Module can be configured, by setting DIP switches, to be addressed as an 8 Input/8 Output, 8 Input, or 8 Output, or 4 Input/4 Output module. If it is configured to be used with both Inputs and Outputs, the module must be installed in a high-density slot in the PLC rack. Multiple modules may be included in a system, as long as the load on the rack power supply is not exceeded, and sufficient I/O references are available.

BASIC Language

The BASIC language for the Series One ASCII/BASIC Module is a subset of the Intel MCS BASIC-52, with additional commands provided by GE Fanuc - NA. These additional commands provide the ability to perform string comparison, calculate string length, renumber BASIC program lines, change baud rate, delete programs, turn LEDs on and off, and reliably transfer data between the CPU and the ASCII/BASIC module. Data can be entered as decimal or hexadecimal numbers (Refer to Appendix D under "Hexadecimal"). *Note that if hexadecimal numbers are to be entered, and they start with A - F, they must be preceded with a 0 (zero), e.g., 0A13CH (H = Hexadecimal).*

Modes of Operation for Memory Use

The ASCII/BASIC Module has two modes of operation for use of memory resources, expanded mode and partitioned mode. ***Use of the expanded mode in conjunction with the ABMHelper program (IC641PBE504) is strongly recommended.*** In the expanded memory mode, all 56K of RAM memory is available for program development and execution. In the partitioned mode the lower 32K of memory is used for program development and execution, and the upper 24K (8000-DFFFH) of memory (which is the archival storage area, or ASA) is used for program storage, similar to using an internal floppy disk. In addition, there is 6K of permanent storage area (PSA) from E000H to F7FFH which can be used as a battery backed storage area for variables which will not be automatically reallocated by BASIC. This means that these variables can be used by other BASIC programs which may be loaded at different times. Normally, loading a new BASIC program will clear all previously declared variables.

The archival storage area is protected by a write protect/enable switch (labeled PROT) located on the faceplate. This provides an option that allows you to protect BASIC programs that have been stored from being overwritten. This switch must be left in the write enabled position for the expanded memory mode. This area is cleared if DIP switch position 3 is in the *Clear on Power Up* position.

The permanent storage area is not write protected with the front panel switch. This area is cleared when DIP switch position 3 is in the clear position on power-up, but is not cleared by a normal power-up cycle. The PSA is used to chain program data or information from one program to the next. If a new program is downloaded, all variable data is lost, except if it has been stored in this area.

On the first power-up, the ASCII/BASIC Module sets the mode to the expanded memory mode. To use the expanded RAM for archival storage, the following command must be executed, either after each clear sequence or in the first line of the program: MTOP = 32767.

NOTE

If the file transfer and storage commands are used without first setting MTOP=32767, unpredictable operation may occur.

To return the module to the expanded memory mode, execute the statement: MTOP=57343.

MCS BASIC-52 Reference Information

The following information will help you in understanding the MCS BASIC-52 programming instructions in this manual.

Argument Stack: The argument stack occupies locations 301 (12DH) through 510 (1FEH) in external RAM memory. This stack stores all constants that MCS BASIC-52 is currently using. Operations such as Add, Subtract, Multiply, and Divide always operate on the first two numbers on the argument stack and return the result to the argument stack. The argument stack is initialized to 510 (1FEH) and "grows down" as more values are placed on the argument stack. Each floating point number placed on the argument stack requires 6 bytes of storage.

Commands: MCS BASIC-52 operates in two modes: Command (or direct) mode and Run (or interpreter) mode. MCS BASIC-52 commands can only be entered when the processor is in the Command mode. This manual uses the terms "Run mode" and "Command mode" to refer to the two different modes of operation.

Constants: A constant is a real number that ranges from ±1E-127 to ±.99999999E+127. A constant can also be an integer. In this manual, constants are referred to as: [const].

Control Stack: The control stack occupies locations 96 (60H) through 254 (OFEH) in external RAM memory. This memory is used to store all information associated with loop control (i.e., DO - WHILE, DO - UNTIL, and FOR - NEXT) and BASIC subroutines (GOSUB). The stack is initialized to 254 (OFEH) and "grows down".

Data Format: The range of numbers that can be represented in MCS BASIC-52 is ±1E-127 to ±.99999999E+127.

There are eight digits of significance in MCS BASIC-52. Numbers are internally rounded to fit this precision. Numbers may be entered and displayed in four formats: integer, decimal, hexadecimal, and exponential. For example: 129, 34.98, OA6EH, 1.23456E+3.

Expressions: An expression is a logical mathematical formula involving operators (both single and dual operand), constants, and variables. Expressions can be simple or quite complex. (For example, 12*EXP(A)/100, H(1)+55, or (SIN(A)*SIN(A)+COS(A))/2.) A "stand-alone" variable ([var]) or constant ([const]) is also considered an expression. In this manual, expressions are referred to as: [expr].

Format Statements: Format statements may only be used within the PRINT statement. The FORMAT statement includes TAB([expr]), SPC([expr]), USING(special symbols), and CR (carriage return with no line feed).

Integers: In MCS BASIC-52, integers are numbers that range from 0 to 65535 or 0FFFFH. All integers can be entered in either decimal or hexadecimal format, and all hexadecimal numbers must begin with a valid digit. (For example, the number A000H must be entered as 0A000H.) When an operator, such as .AND. requires an integer, MCS BASIC-52 will truncate the fraction portion of the number so it will fit the integer format. All line numbers used by MCS BASIC-52 are integers. In this manual, integers and line numbers are referred to, respectively, as: [integer] - [ln num].

NOTE

Throughout this document, the brackets [ ] are used only to indicate an integer, constant, etc. They are *not* entered when typing the actual number or variable.

GFK-0249

Internal Stack: The stack pointer on the 8052AH microcontroller (special function register, SP) is initialized to 77 (4DH). The 8052AH's stack pointer "grows up" as values are placed on the stack. In MCS BASIC-52 you have the option of placing the 8052AH's stack pointer anywhere (above location 77) in internal memory.

Line Editor: MCS BASIC-52 contains a minimum-level line editor. Once a line is entered, you may not change the line without retyping it. However, you may delete characters while entering a line by using the RUBOUT or DELETE character (7FH). The RUBOUT character will cause the last character entered to be erased from the text input buffer.

Additionally, the Ctrl D key combination will cause the entire line to be erased. Ctrl Q (X-ON) and Ctrl S (X-OFF) recognition has been added to the serial port. However, you must be careful not to accidently type a Ctrl S when entering information because the MCS BASIC-52 will no longer respond to the console device. Ctrl Q is used to bring the console device back to life after typing Ctrl S.

Operators: An operator performs a pre-defined operation on variables and/or constants. Operators require either one or two operands. Single operand operators are SIN, COS, and ABS. Dual operand operators include Add (+), Subtract (-), Multiply (*), and Divide (/).

Relational Expressions: Relational expressions involve the operators EQUAL (=), NOT EQUAL (< >), GREATER THAN (>), LESS THAN (<), GREATER THAN OR EQUAL TO (>=) and LESS THAN OR EQUAL TO (< =). They are used in control statements to "test" a condition (i.e., IF A<100 THEN ...). Relational expressions *always* require two operands. In this manual, relational expressions are referred to as: [rel expr].

Special Function Operators: Virtually all of the special registers on the 8052AH microcontroller can be accessed by using special function operators. The exceptions are PORTS 0, 2, and 3, and non-I/O associated registers such as ACC, B, and PSW. Other special function operators are XTAL and TIME.

Stack Structure: MCS BASIC-52 reserves the first 512 bytes of external data memory to implement two "software" stacks. These are the control stack and the arithmetic/argument stack. Understanding how the stacks work in MCS BASIC-52 is *not* necessary if you wish to program only in BASIC. However, understanding the stack structure *is* necessary if you wish to link MCS BASIC-52 to assembly language routines.

Statements: A BASIC program is comprised of statements. Every statement begins with a line number, is followed by the statement body, and then terminates with a carriage return (cr), or a colon (:) in the case of multiple statements per line. Some statements can be executed in the Command mode; others cannot. There are three general types of statements in MCS BASIC-52: Assignments, Input/Output, and Control.

- Every line in a program must have a statement line number ranging between 0 and 65535, inclusive.
- Statement numbers are used by BASIC to order the program statements sequentially.
- In any program, a statement number can be used only once.
- Statements need not be entered in numerical order, because BASIC will automatically order them in ascending order.
- A statement may contain no more than 79 characters.
- Blanks (spaces) are ignored by BASIC, and BASIC automatically inserts blanks during LIST.

- More than one statement can be put on a line, if separated by a colon (:), but only one statement number is allowed per line.

System Control Values: System control values include the following: LEN (which returns the length of the program), FREE (which designates how many bytes of RAM are not used that are allocated to BASIC), and MTOP (which is the last memory location that is assigned to BASIC).

Variables: A variable can be defined as either:

- A letter (i.e., A, X, I).
- A letter followed by a number (i.e., Q1, T7, L3).
- A letter followed by a one-dimensioned expression (i.e., J(4), G(A+6), or I(10*SIN(X))).
- A letter followed by a number followed by a one-dimensioned expression (i.e., A1(8), P7(DBY(9)), or W8(A+B)).

Variables may also contain up to 8 letters or numbers, including the underline character. This allows you to use a more descriptive name for a given variable. Examples of valid variables in MCS BASIC-52 are: FRED VOLTAGE1 I_I1 ARRAY(ELE_1).

It should be noted that only the *first character*, the *last character*, and the *length of the variable* are significant. Therefore, the variables "John", "Jean", and "Joan" will be assigned the same values.

When using the expanded variable names it is important to note that it takes longer for MCS BASIC-52 to process these expanded variable names. Also, you may not use any keyword as part of a variable name. (For example, the variables TABLE and DIET could not be used because TAB and IE are reserved words.) BAD SYNTAX errors will be generated if you attempt to define a variable with a reserved word.

Variables that include a one-dimensioned expression ([expr]) are often referred to as "dimensioned" or "arrayed" variables. Variables that only involve a letter or a letter and a number are called "scalar" variables. (Refer to the description of the DIM statement for more information.) In this manual, variables are referred to as: [var].

NOTE

It takes MCS BASIC-52 less time to find a scalar variable because there is no expression to evaluate in a scalar variable. To make your program run as fast as possible, use dimensioned variables only when you have to. Use scalars for intermediate variables, and then assign the final result to a dimensioned variable.

MCS BASIC-52 allocates variables in a "static" manner. That means each time a variable is used, BASIC allocates a portion of memory (8 bytes) specifically for that variable. This memory cannot be deallocated on a variable-by-variable basis. If you execute a statement like Q=3, you cannot later decide that the variable Q no longer exists. The only way you can clear the memory that is allocated to variables is to execute a CLEAR statement. This statement frees all memory allocated to variables.

Chapter 2
Installation and Configuration

An ASCII/BASIC Module can be installed in any I/O slot in a Series One or One Plus CPU rack or I/O expansion rack. If the module is to be configured for use with both inputs and outputs, it must be installed in a high-density slot (addressable for 16 I/O references). Multiple modules can be installed in a rack, as long as the power requirements do not exceed the current supplied by the power supply in the rack. Current requirements, expressed as units of load, for all modules in a rack must be added to calculate total current requirements. Refer to the Series One/One Plus User's Manual, GEK-90842, for further information on units of load for individual modules.

ASCII/BASIC Module Hardware Features

Hardware features of the ASCII/BASIC Module are described in this paragraph. A group of seven LED status indicators, a memory protect switch, and a 9-pin D sub-miniature connector are mounted on the faceplate. Two LEDs are user definable through BASIC programming. The 9-pin connector provides access to the RS-232 port for connection to a programming device, or to a user supplied serial device. The memory protect switch provides write protection for the archival storage portion of RAM memory. A 4-position DIP switch located on the lower back of the module is used to configure selectable operating parameters. The lithium battery (catalog number IC610BAT100) for backing-up the volatile RAM memory is mounted on the smaller of the two circuit boards in the module. The battery is replaceable by the user, and is monitored by circuitry in the module to ensure that the voltage of the battery is at the proper operating level to maintain the RAM memory during a no-power condition.

Status Display

Seven LEDs provide a visual indication of system operation and diagnostic information.

Table 2-1. Status Indicator Definitions

LED	Color	Definition
PWR	Green	ON = Power is being supplied at the proper level by the rack power supply. OFF = Incorrect or no power being supplied to this module.
DIAG	Red	On = This module is not operating properly (internal module error). OFF = Module is operating properly.
BATT	Red	ON = Voltage of battery is low, or no battery present. OFF = Battery present and at the proper voltage level.
RXD	Amber	ON = Data is being received by this module. OFF = No data is being received.
TXD	Amber	ON = Data is being transmitted by this module. OFF = Data is not being transmitted by this module.
USR1	Amber	Can be user defined in BASIC program. LED0 1 command turns USR1 off. LED1 1 command turns USR1 on. This LED also reflects the status of the RTS output signal.
USR2	Amber	User defined in BASIC program. LED0 2 command turns USR2 off. LED1 2 command turns USR2 on.

As part of the power-up diagnostic for the module, each time that a power cycle occurs, all of the LEDs will turn on for approximately one second. After the one second duration all LEDs, except for the PWR LED, will turn off. While the LEDs are on during this cycle, the ASCII/BASIC Module will not respond to the space bar for autobaud operation on the programming device. Autobaud operation is described below.

RS-232 Serial Communications Port

Access to the RS-232 serial port on the ASCII/BASIC Module is through the 9-pin D-subminiature connector mounted on the lower part of the faceplate. This port is used for both programming and user interface, and supports the XON, XOFF protocol. The port is not ground isolated from the rack backplane.

The data format for communications with this module is: start, 1 bit /Data, 8 bits/Parity, none/Stop, 1 bit. This port supports RTS/CTS. Baud rate is selectable from 300 to 19,200 baud, and is determined at power-up by sending a 20H (space bar on programmer keyboard) character to the module, or previously programming it in the resident BASIC program. Baud rate can also be set with the PORTSET baud command in the BASIC program. This eliminates the need to press the space bar and use the autobaud routine on the next power-up. For initial start-up after clearing the module, the space bar is pressed.

NOTE

RTS is not supported in version A (catalog number IC610ABM100A) of this module. Modem applications requiring RTS/CTS can usually be handled with the converter/repeater unit (Adaptor Unit, IC630CCM390) when using a version A ASCII/BASIC Module.

a42678

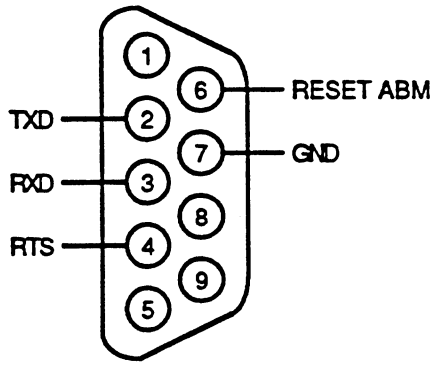


Figure 2-1. RS-232 Communications Port Configuration

Table 2-2. Serial Port Pin Definitions

Pin	Signal Name	Description
2	TXD	Transmit Data
3	RXD	Receive Data
4	RTS	Control Output
6	RESET ABM	Resets ABM
7	GND	Signal ground (0V)
Frame	DC COMM	DC common, Connector Shell

The RTS Signal

The ASCII/BASIC Module has an RS-232 compatible RTS output which can be used in conjunction with modems and other devices which require a "handshake" signal. This feature allows the use of the ASCII/BASIC Module in multidrop applications with modems, and with certain other devices. The A revision of this module does not include the RTS feature.

The RTS output signal is available on pin 4 of the serial port connector. The RTS signal is controlled by the LED1 statement in the BASIC program, and the USR1 LED on the front panel shows the status of the RTS signal. If the LED is on, the RTS signal is in its "true" state.

If it is not necessary to use the RTS capabilities, then the USR1 LED can be used as a general status LED from BASIC program control.

Using the RTS Signal

The recommended procedure for using RTS is as follows:

1. Turn on RTS using the LED1 1 command.
2. Delay a period of time depending on your application. This time can usually be determined by experimental means (e.g. start with a short delay time, and keep increasing it until it works OK, then add 50% more). For most devices, this time would be less than 1 second.
3. Use the PRINT statement to send the data from the port.
4. Turn off RTS using the LED1 0 command.
5. Use the INPUT, GET, or INPB statements to receive data, if required.
6. Loop back and do it again.

Example of an Application Using RTS

A typical application using RTS is as follows:

Port configuration using RTS.

```
ABM (Revision B)                    Modem or other device

Pin
 2 ----------------------------> Received Data
 3 <---------------------------- Transmitted Data
 4 ----------------------------> RTS or CTS Input
        no connection X-<-----   RTS or CTS Output
 7 ----------------------------- Ground
```

BASIC Program Example:

```
1000 REM This program sends "Hello" to the remote device, then waits for
1010 REM a 3 character response from the remote end, which is printed.
1015 STRING 100,10      : REM Save string space for the INPB below.
1020 LET FLAG = 99      : REM This variable must be declared before it is used.
1030 LET DELAY = .5     : REM .5 seconds seems to work OK in this application.
1040 CLOCK 1            : REM This starts up the clock in BASIC.
1045 REM This is the end of the setup section.
1047 REM The main loop follows.
1050 LED1 1             : REM This turns on RTS, and the USR1 LED.
1060 GOSUB 2000         : REM This is a delay to wait for CTS.
1070 PRINT "Hello"      : REM This goes out the serial port to the remote.
1080 LED0 1             : REM This turns off RTS, and the USR1 LED.
1090 REM The statement below waits 3000 ms to receive 3 characters in $(1).
1100 INPB $(1),3,3000,FLAG
1110 PRINT $(1)         : REM Printout the response from the other end.
1120 GOTO 1050          : REM Loop back to do it again.
1130 REM This is the end of the main program.
1140 REM
1150 REM
2000 REM This is a delay to wait for CTS.  This value can be experimentally
2010 REM determined in any application.
2020 LET DBY(71) = 0            : REM Reset the fractional portion of TIME.
2020 LET OLDT = TIME            : REM Get the time the loop was entered.
2030 LET NOWT = TIME            : REM Store away the present time.
2040 LET DELT = NOWT - OLDT     : REM Calculate how long in the loop so far.
2050 IF DELT >= DELAY THEN RETURN   : REM If delay time is reached, exit.
2060 GOTO 2030                      : REM If not, then keep looping.
2070 REM This is the end of the delay subroutine.
```

GFK-0249

DIP Switch Configuration

The 4-position DIP switch on the lower back of the module must be configured at installation. This is the only item requiring configuration by the user. Before installing an ASCII/BASIC module in the selected I/O slot, configure the DIP switch to the required parameters as shown below.

Table 2-3. DIP Switch Configuration and Definition

<table>
<tr><th>Position</th><th colspan="3">Definition</th></tr>
<tr><td>1 and 2</td><td colspan="3">Select I/O Points used by Module</td></tr>
<tr><td></td><td>Switch 1
ON
OFF
ON
OFF</td><td>Switch 2
ON
ON
OFF
OFF</td><td>I/O Points
8 In/8 Out
8 In
4 In/4 Out
8 Out</td></tr>
<tr><td>3</td><td colspan="3">ON = On power-up, the ASCII/BASIC module goes to RUN mode and executes the program in development memory, or optionally executes a program from the Archival Storage Area. To make program changes, abort the RUN mode using the Ctrl/C key sequence. ABMhelper automatically aborts RUN mode on power-up. If the Run/Trap mode is enabled, program execution cannot be aborted with Ctrl/C.

If you do not want to run the program immediately after power-up, the first statement in the program should be a STOP command. This will not allow the program to execute automatically on power-up. The STOP command can be removed after program development is completed.

OFF = On power-up, clears the entire BASIC program, variable memory, the Archival Storage Area, and the Permanent Storage Area. This switch should be placed in the OFF position only for initial start-up, or if it is desired to clear a program from memory when the Run/Trap mode is enabled and a password has accidentally not been programmed. At the end of the clear sequence, the module goes into the autobaud mode, waits for a space bar from the terminal device, then prints the message, ABM HAS BEEN CLEARED, RETURN TO NORMAL MODE. This is a reminder to return DIP switch position 3 to the ON position. You can leave the switch in the OFF position, however memory will be cleared after each power-up.</td></tr>
<tr><td>4</td><td colspan="3">ON = Normal Operation

OFF = Diagnostics enabled. If the module is configured for the 8 Inputs mode, the self diagnostics routine will be performed. These diagnostics consist of the following items:
• System ROM check
• Internal RAM check
• External RAM check
• Loop back check for communications port
This test requires that pins 2 and 3 of the RS-232 communications port be connected together.
Failures detected by the diagnostics are indicated as:
• All LEDs on = System ROM checksum error
• DIAG and POWER on = Internal system RAM error
• All LEDs blinking = External system RAM error
• USER1 and USER2 LEDs on = Serial port error
• No LEDs on = No errors</td></tr>
</table>

The factory defaults for the DIP switch settings are for all switches set to the ON position. If you should encounter any start-up problems with the ASCII/BASIC Module, set DIP switch position 3 to the OFF position. This will clear memory when power is next turned on. Then, turn power off and reset the switch to the ON position for normal operation. When the module is powered-up, the PORTSET statement should be used to set the working baud rate, then press the Enter (or Return) key.

Run/Trap Mode

A feature called the Run/Trap mode is available which provides a method for the user to protect programs from unauthorized viewing, and/or modification by unauthorized personnel. For example, if you have written a unique program intended for use in multiple systems and you do not want unauthorized access to the program. For those programs you wish to protect, program security can be assured by use of the Run/Trap mode. This mode is initiated by writing A5H to FFFAH, and writing 5AH to FFFBH with the commands, LET XBY(0FFFAH)=0A5H, and LET XBY(0FFFBH)=05AH. ***Be sure to have an "escape password" programmed, and save your program before doing this step.***

On the next power-up cycle, if DIP switch position 3 is in the ON position (normal operation), the ASCII/BASIC Module will go into the Run/Trap mode. It is then not possible to exit the program by using the Ctrl/C key sequence as you can during normal operation with the program unprotected. You also cannot list or upload the program. The program stays in the Run mode. A custom password must be built into the program which writes 0 to FFFA,BH. Then Ctrl/C will be re-enabled after the next power cycle. If the password is forgotten, you must totally clear the program by initiating a power-up cycle with DIP switch 3 in the OFF (clear) position. Use of the Run/Trap feature is not recommended unless it is absolutely required.

Memory Usage For Program Development

In the previous chapter, the two methods of operation using available memory, which are expanded mode and partitioned mode, were discussed. Before developing your programs, you must decide which method to use. Use the criteria described below as a guide for choosing which method to use.

Expanded Mode of Operation

This method uses all of the ASCII/BASIC Module's memory for development of a single program. The program is developed using the full 56K of available memory. If it is desired to save revisions to the program during development, it is necessary to use a computer with the ABMHelper program installed to retrieve and save the revisions. This method allows the development of programs which require the full 56K of memory. If this method is used, there is no Archival Storage Area (ASA), and the BURN, RAM, ROM, RROM, XFER commands are not used. This is the simplest and most straightforward method to use.

Partitioned Mode of Operation

This method splits the available memory into a 32K RAM development area, and a 24K Archival Storage Area (ASA). Multiple programs can be stored in the ASA, then retrieved to the 32K RAM area, and executed. With this method, the total program size for all the archived programs is 24K, and the total size available to the executing program and variables is 32K. The programs to be archived are developed in RAM, then saved to the ASA with the BURN command.

Programs in the Archival Storage Area are deleted by the DELPRM command. The DELPRM command deletes a specific program, then compresses the remaining programs to allow for more program storage.

This mode of operation is valuable if you are using a dumb terminal for programming. Partitioned mode is enabled by sending the command MTOP=32767 to the ASCII/BASIC Module. If you are using a dumb terminal as your programming device, this mode should be considered.

Use of Keywords

When writing your BASIC program, none of the following keywords may be used as part of a variable name, even though the keyword is listed as *"not supported"* or *"do not use"*. The BASIC commands which are supported by the Series One ASCII/BASIC Module are listed below. Refer to the applicable chapter in this manual for detailed information on using each of the keywords.

Table 2-4. BASIC Commands Supported in All Modes

Commands	Statements	Operators
RUN	CLEAR	ADD (+)
CONT	CLEARS	DIVIDE (/)
LIST	CLEARI	EXPONENTIATION (**)
NEW	CLOCK1	MULTIPLY (*)
NULL	CLOCK0	SUBTRACT (-)
	DATA	LOGICAL AND (.AND.)
	READ	LOGICAL OR (.OR.)
	RESTORE	LOGICAL X-OR (.XOR.)
	DIM	LOGICAL NOT
	DO-WHILE	ABS ()
	DO-UNTIL	INT ()
	END	SGN ()
	FOR-TO-STEP	SQR ()
	NEXT	RND
	GOSUB	LOG ()
	RETURN	EXP ()
	GOTO	SIN ()
	ON-GOTO	COS ()
	ON-GOSUB	TAN ()
	IF-THEN-ELSE	ATN ()
	INPUT	=, >, >=, <, <=, <>
	LET	ASC ()
	ONERR	CHR ()
	ONTIME	CBY ()
	PRINT	DBY ()
	PHO.	XBY ()
	PH1.	GET
	PUSH	TIME
	POP	XTAL
	REM	MTOP
	RET1	LEN
	STOP	FREE
	STRING	PI
	LD@	
	ST@	
	IDLE	

Commands Used Only in Partitioned Mode

The following commands should only be used when operating in the partitioned mode.

```
XFER
RROM
ROM
RAM
DELPRM
BURN
```

GE Fanuc - NA Custom BASIC Commands

In addition to the BASIC commands provided by Intel BASIC-52, GE Fanuc - NA has developed several other useful ones. A brief description of these commands is provided below. For a detailed description of these commands, see Chapter 6 in this manual.

Table 2-5. GE Fanuc - NA Custom BASIC Commands

Command	Function of Command
TRANSFER	Exchanges data with the Series One CPU.
LSTR	Calculates the length of a string.
SPLIT	Reads a single bit or field of bits.
MOVE	Sets a single bit or field of bits.
SWITCH	Read the DIP switch settings on the ASCII/BASIC Module. This allows the program to monitor the board settings.
MATCH	Does a String comparison.
INPB	Fetches specified number of characters from the serial port and stores them in the specified string.
RENUM@	Renumber the BASIC line numbers.
DELPRM	Deletes a program stored in the Archival Storage Area.
PORTSET	Selects the baud rate for data transfer between the module and the PLC. Selections are 300, 600, 1200, 2400, 4800, 9600, or 19200 baud.
LED0 1 or 2 *	Turns the specified LED off.
LED1 1 or 2 *	Turns the specified LED on. Also used for RTS control (see page 2-3)

* These 2 commands enable the two USER LEDs on the front panel to be turned on and off under program control.

Sign On Message

On power-up, after the internal diagnostics are complete and if no program is in memory, the first display you will see on the screen is the sign-on message:

```
Welcome to GE Fanuc BASIC Vx.x
```

Vx.x is the applicable version number of BASIC that is resident in your ASCII/BASIC Module.

GFK-0249

Commands, Statements, and Operators Not Supported

The following BASIC commands, statements, and operators should not be used, since they are not supported by the Series One ASCII/BASIC Module.

```
LIST#, PRINT#, PH0.#, PH1.#
PGM
PROG
PROG1 to PROG6
FPROG1 to FPROG6
BAUD
ONEX1
```

The following statements, commands, and operators are for use only when writing custom assembly language routines, and are not normally used. If you wish to use these functions, it is recommended that you obtain a copy of the Intel manual, catalog number 270010-002 or later. This manual contains detailed information on these items.

Command	Function of Command
CALL	Used to call an assembly language routine.
LIST@	Requires a custom driver to be written in assembly language.
PRINT@	Requires a custom driver to be written in assembly language.
PH0.@	Requires a custom driver to be written in assembly language.
PH1.@	Requires a custom driver to be written in assembly language.
U1	Requires a custom driver to be written in assembly language.
UO	Requires a custom driver to be written in assembly language.

IE, IP, PCON, T2CON, TCON, TMOD, TIMER0, TIMER1, TIMER2: These require a detailed technical knowledge of the 8052 chip, and should not be used.

Troubleshooting the ASCII/BASIC Module

The ASCII/BASIC Module is designed to provide trouble free operation for many years. If any problems should occur during operation of the module, you can easily troubleshoot it by observing the status display, which is the main troubleshooting tool. The status display consists of seven LEDs on the front of the module. Definitions for the ON and OFF condition for each LED can be found in table 2-1.

Note that all of the status indicators are relatively self-explanatory. A basic troubleshooting guide is provided for you in the following paragraphs.

Troubleshooting Guide

Each of the status indicators is listed below. The normal status of each LED is noted along with some possible cures for the problem causing an abnormal indication.

- **PWR**: Normal condition - ON. If this LED is OFF, check the voltage source to the rack. If it is correct, the problem is either a faulty power supply, or the ASCII/BASIC Module.
- **DIAG**: Normal condition - OFF. If this LED is ON, it indicates that an internal diagnostic has failed. The ASCII/BASIC Module should be replaced.
- **BATT**: Normal condition - OFF. When this LED turns ON, the lithium back-up battery for RAM memory is either dead, or its operating voltage is too low to maintain the contents of RAM memory in

the event of a power loss. When this condition occurs, the battery should be replaced within one week.

- **RXD and TXD**: These LEDs reflect the communication status between the ASCII/BASIC Module and the Series One PLC. When ON, data is being received by (RXD) or transmitted from (TXD) the module. If communication is attempted and the LEDs do not turn on, or if the LEDs stay on continuously even though the ASCII/BASIC Module is not communicating, check for proper connection of the cable to the module, or a damaged cable. If the cable and its connections are good, replace the module.
- **USR1 and USR2**: The condition or status of these LEDs is defined by the user's program. If they are not providing a status as expected, check your program, or if a physical device is being monitored, check that device.

Replacing the Lithium Battery

- Before you proceed with the following steps, have a new lithium battery (IC610BAT100) available for immediate replacement.
- Save a copy of the program to diskette using ABMHelper, if possible.
- Turn off power to the rack in which the module is installed.
- Remove the module from its slot.
- Refer to the following figure for location of the battery on the ASCII/BASIC module.

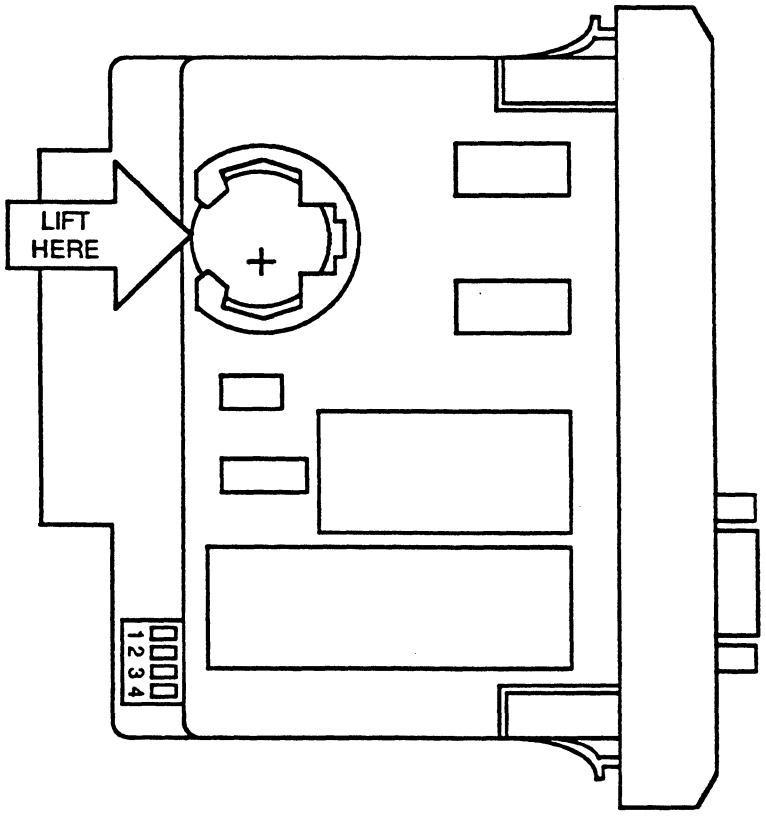


Figure 2-2. Location and Replacement of Lithium Battery

GFK-0249

- To remove the battery, pull up on the part of the battery not covered by the battery holder.
- Once the battery has been removed from the module, it must be replaced within thirty minutes to ensure that the contents of RAM memory will not be lost.
- Insert the new battery by placing it in the holder and pushing down until it is seated in place.
- Reinstall the ASCII/BASIC Module into its slot in the rack.
- Apply power to the rack and observe the LEDs for proper operation.
- Reload the program previously saved on diskette (if required).

Chapter 3
Creating and Storing BASIC Programs

GFK-0249

When using the Series One ASCII/BASIC Module, the following functions typically need to be performed:

- Creation of the user's BASIC program
- Storage of the user's BASIC programs from the ASCII/BASIC Module memory to disk
- Retrieval of the user's BASIC programs from disk to the ASCII/BASIC Module's memory

A program called ABMHelper has been developed to assist in implementing the functions listed above. In addition, many commercially available packages (such as VTERM) are available which will also perform these functions.

Advantages of Using ABMHelper

Using the ABMHelper program to assist in implementing the ASCII/BASIC Module's features provides the following advantages:

- Provides terminal emulation capability for direct connection to the ASCII/BASIC Module.
- Provides program storage and retrieval functions using the computer's diskette functions.
- Provides a full screen editor. This is also useful for non-ASCII/BASIC Module editing tasks.

The following procedures are recommended for use with the Series One ASCII/BASIC Module if you will be using an IBM-PC, or compatible computer

- Use ABMHelper running on a Workmaster computer, or other IBM-PC compatible computer.
- Use the ASCII/BASIC Module in Expanded Memory Mode. This will allow all 56K of RAM to be used for program development and execution. Preliminary versions of the developing program can be saved to disk as you proceed.
- Edit your program with the W-ED full screen editor, using either the "Source" or "No Source" operation described below.

If you will be using a dumb terminal with the ASCII/BASIC Module, the following procedures are recommended:

- Use the dumb terminal as the interface device.
- Use the ASCII/BASIC Module in *Partitioned Memory Mode*. By using this mode, preliminary versions of the program can be saved in the ASCII/BASIC Module's Archival Storage Area.
- Edit the program using the built-in line editor.
- Add remarks to the code in the form of numbered lines, e.g., *"1000 REM This is the start of the PID algorithm"*.

SOURCE/NOSOURCE Configuration

ABMHelper has several different edit modes. Depending on your own circumstances, you may have a strong preference for one mode or another. You must select either the SOURCE or NOSOURCE edit mode. The operation of ABMHelper is different depending on which mode you have selected. Normally, once you have chosen this mode, it will not be changed again. Within both SOURCE and NOSOURCE modes, there are two edit sub-modes which may be used under different circumstances.

These modes are also described below. The default setting is NOSOURCE (Ctrl/E and Ctrl/F are active - refer to Chapter 4, *ABMHelper* for detailed information).

NOSOURCE Mode of Operation

The NOSOURCE mode of operation assumes that the user wishes to develop programs in the following manner:

1. Enter and edit programs using ALL of the following:
   A. The built in ASCII/BASIC Module hardware line editor. To change the program, type in a new line which replaces the old line.
   B. The Ctrl/E edit function, which loads the edit buffer with a small portion of the program, then merges the edited file back into the ASCII/BASIC Module hardware. The W-ED full screen editor is used. The reloading process can be very fast since only a small portion of the program is edited and reloaded. To delete a line from the editor it is necessary to have the line number followed by a blank line.
   C. The Ctrl/F edit function, which loads the edit buffer with the entire program, then clears the ASCII/BASIC Module hardware before reloading the edited file. The W-ED full screen editor is used.
2. Save program revisions to diskette using the "Save program to disk" function selected as an option from the ABMHelper main menu.
3. Program commentary is an actual part of the program using numbered program lines which contain REM statements.
4. Program documentation aids such as indentation of FOR-NEXT loops, and addition of spaces between words is lost every time the program is downloaded to the ASCII/BASIC Module.

SOURCE Mode of Operation

The SOURCE mode of operation assumes that the following development methodology is used:

1. Programs are edited using the W-ED editor supplied with ABMHelper.
2. Program revisions are saved to diskette automatically as part of the edit process.
3. After each program edit, the entire program is downloaded to the ASCII/BASIC Module. This could take several minutes, even though the changes to the program were minor. There is a way to download only a portion of the program to save time (see *"Quick" SOURCE Mode Editing).*
4. If the ASCII/BASIC Module program is changed using the built-in line editor (which is convenient for testing), these changes cannot be uploaded to the source file. It is necessary to re-enter the changed lines in the source file, and download the entire program again.
5. Program documentation can include remarks without line numbers. These comments are not loaded with the program, and do not take up space in the ASCII/BASIC Module (they are ignored by the module when the program is downloaded).
6. Program documentation aids such as indentation of FOR-NEXT loops are retained.

The main advantage of the NOSOURCE mode is that changes can be made quickly and tested quickly. The "master" program is actually in the ASCII/BASIC Module hardware. The main advantage of the SOURCE mode is that programs can be clearly documented with no penalties. The "master" program is on the disk.

Since some ASCII/BASIC Module programmers may be familiar with one particular style and have a strong preference for that style, both mechanisms have been provided.

SOURCE/NOSOURCE Sub Modes

As previously described, there are four different ways to edit.

1. NOSOURCE mode with Ctrl/E (recommended for long programs, experienced programmers)
2. NOSOURCE mode with Ctrl/F (recommended for short programs, novice programmers)
3. SOURCE mode with quick edit (recommended for long programs, experienced programmers)
4. SOURCE mode with no quick edit (recommended for short programs, novice programmers)

The SOURCE/NOSOURCE mode decision should be made first, then the sub-mode chosen based on program length and familiarity with the "merge" concept described below.

For novice users, and when working with short programs, it is recommended that either method 2 or 4 be used. In both cases, the entire program is loaded into the editor, and the whole edited program is downloaded back into the ASCII/BASIC Module after the ASCII/BASIC Module's memory is cleared. This is the safest method to use, although it takes considerable time to download a long program (several minutes). Basically, what you see on the editor screen is what you get in the ASCII/BASIC Module after downloading. Conceptually, these are the easiest methods to understand. These methods are also preferable when working with short programs, since the download time is not significant.

Both methods 1 and 3 "merge" only a part of the program back into the ASCII/BASIC Module. The previous program is not cleared. This saves considerable download time, but the user must understand the concept of merging. For example, it is not possible to delete a line in the editor, download the now non-existent line, and have the line disappear in the ASCII/BASIC Module. The merge operation basically overwrites the present program, and leaves untouched any lines which are not replaced by the new edited file. If you are familiar with the GW-BASIC or BASICA editors, then you would probably also be comfortable with these methods. These methods become valuable when long programs are being edited, and only small changes are being made.

NOTE

If you still are unsure of which mode to use, then the default setting of NOSOURCE, used in conjunction with the Ctrl/F function is recommended. *Refer to Chapters 4 and 5, in this manual, for more information about editing your ASCII/BASIC Module programs.*

"Quick" SOURCE Mode Editing

After editing of the source file has been completed, you have two options regarding reloading the ASCII/BASIC Module.

1. By exiting the edit mode with the F1 key (to save the changes), then pressing any key except Esc at the prompt, the entire source file will be transmitted to the ASCII/BASIC Module. This can take considerable time if the file is long (even if only minor changes were made), but this method is easier to understand.
2. An alternative method exists, but is more complicated. If only a few lines are changed in the source file, and these lines are relatively close together, the following procedure can be used to minimize the download time to the ASCII/BASIC Module.
   A. Finish the editing to be done to the source file.
   B. Using the Alt L key sequence, mark the beginning and end of the lines that you changed.
   C. Press the Esc key to place the cursor on the command line. Then, type WED Z and press the Enter key. A blank screen will be displayed.
   D. Press the Ctrl/B keys. This will import the marked lines from the main source file.

E. Press the F1 key, then the Enter key. This will return you to the main source file after creating a file named Z which includes the changes you made.

F. Press the F1 key, then the Enter key to save the new source file to disk. Press the Enter key at the prompt to merge the Z file into the ASCII/BASIC Module.

This method can save considerable time if you are working with a large source file. It should be noted, however, that you cannot delete a line in the ASCII/BASIC Module by eliminating it in the source file, then merging the result as described above. The line number must be retained with a blank line following it.

NOTE

"Quick" SOURCE mode editing is recommended only for those who are familiar with merging of programs. Refer to the previous discussion of SOURCE/NOSOURCE sub-modes.

Configuring ABMHelper (Revision 1.02 and later)

Version 1.02 (and later versions) of ABMHelper provides a mechanism to change the operating parameters of ABMHelper. The setup parameters are contained in the ABM.PSU file on the distribution diskette. The parameters which can be changed are :

1. Default baud rate
2. Default communications port
3. Default time/date query on entry to ABMHelper
4. Edit mode

The default settings for these parameters are:

```
9600 baud
Communications port = COM1
Time and date query = ON
Edit mode is NOSOURCE (Ctrl/E and Ctrl/F activated)
```

The edit mode selection was previously discussed.

The ABM.PSU file also contains some additional elements which are not changed by the user. The user can change the ABM.PSU default settings by pressing the B key while in the main ABMHelper menu. This activates the interactive setup function. After questions are answered regarding the above items, a new ABM.PSU file is automatically created on disk, and read back into ABMHelper.

Baud Rate Setup

The default baud rate setting can be 1200, 2400, 4800, or 9600 baud. 9600 baud is preferred for all ABMHelper operations since all serial transfers will occur more quickly at this baud rate. ABMHelper cannot operate at 19200 baud although the ASCII/BASIC module can. If the Series One ASCII/BASIC Module is put into the 19200 baud mode with the *PORTSET 19200* command (to interface with a different device), the only way to re-establish communications with ABMHelper is to clear the module using DIP switch 3, re-establish communications at a lower baud rate, then reload the program.

Communications Port Setup

The default communications port is set to COM1. It is necessary that the computer running ABMHelper has its ports set to work properly. If COM1 is not available on your computer, COM2 is the alternate setting.

Time and Date Setup

The Time and date query on entry to ABMHelper is defaulted to ON. If you have an on-board real-time clock in your computer, and your AUTOEXEC.BAT file has been modified to read this clock on computer power-up, you may want to disable this automatic query. If your initial power-up date setting keeps coming up 1980, you may want to be reminded to set the correct date on entry to ABMHelper. This date is used to time stamp all files saved to disk from ABMHelper, or any other program. Even if the automatic query is disabled, it is still possible to set the time and date from ABMHelper by selecting option 8, *Set/view time and date*, from the Main Menu.

Hints for Documenting BASIC Programs

When writing programs in BASIC, there are some rules that should be followed for efficient use of memory and program structure. An excellent source of information, including many examples, on programming in BASIC is *"The Little Book of BASIC Style"* by John Nevinson, available from Addison-Wesley.

During the early years of Personal Computers many "tricks" were required to conserve memory space. Memories of 2K or 4K bytes were standard, and long programs could not be contained in these memories. These tricks mainly consisted of removing spaces from between words, no comments for the code, and leaving no white space between lines of executing code.

Although these techniques had the desired effect of minimizing code space, they also had the undesirable effects of making the code hard to develop and even more difficult to maintain.

With the new developments in memory products, it is generally no longer necessary to use these tricks, since memory capacities are large enough to handle the code, even when extensively documented.

We strongly recommended that your BASIC programs use the following guidelines. It will save you time and money in the long run, and avoid the frustration of finding your way through your programs, especially the long ones.

- ***Use only one basic statement per line of code.*** This is the most critical item in this list.
- Use remarks liberally, especially if you are using the SOURCE editing mode. The use of remarks for good program documentation cannot be over emphasized. For example, you know exactly what this code does now, but return to it after a two week vacation (or worse, after 2 years) and see how much detail of the program you remember!! Also, if you are creating and editing your program exclusively in SOURCE mode you should be aware that remarks (REM) without line numbers are ignored by the ASCII/BASIC Module when the program is loaded. They do not use any memory space in the ASCII/BASIC Module.
- When using the SOURCE editing mode, use spaces in lines to make the lines more readable. Don't squeeze words together. Make your program readable, so that you or others can understand it.
- Separate sections of code which have a common purpose with white space. It makes it easier to spot when you need to revise that section.
- If you are are using the SOURCE editing mode, it is recommended that you use indentation when programming IF-THEN-ELSE, DO-WHILE loops, and similar instructions. When you are debugging

your program and trying to figure out why it doesn't work, it's nice to be able to quickly find where the end of the loop is located.

- Separate remarks that you enter from code with white space, and put remarks on separate lines.
- Use line number values of equal length. For example, use 1000 through 9999, not 1, 10, 100, 1000.

General Programming/Debug Hints

The following information is provided to aid you in programming and debugging your BASIC programs.

1. Document the program liberally as previously described.
2. If you are developing a menu driven program (similar to ABMHelper), you may want to write the menu handler portion of the program first, with the menu selections branching off to routines which have not yet been written. This demonstrates the overall "user interface" of the program, and provides a framework to build on.
3. Code and test small portions of the program before they are added to larger parts of the program.
   - Code the new routine using the ABMHelper editor.
   - Insert a temporary STOP instruction at the end of the new routine. This is later removed when the program segment has been tested.
   - Set variables, etc. to the correct state in "direct mode" (for example, LET X = 27), then execute the new routine with a GOTO XXXX instruction, where XXXX is the start of the new routine.
   - After the program portion has executed, check the status of variables, strings, etc. that were to be changed in the program. For example, one way to do this is to type *PRINT X* at the command prompt.
   - Make changes using the editor, save the changes to disk, then go back two steps to "Set variables, etc. .....". Or, proceed to the following step.
   - Remove the STOP instruction, and test the program segment as part of the overall program.
4. Test your program with all possible values. For example, if your program prints a menu that prompts *"Select 1, 2, 3, 4"*, what happens if the response is 5 or Ctrl/N, or ABCD?
5. If you are developing a program in a system that experiences frequent power cycles, you may want to add a STOP statement in the first line of the program. This STOP statement can be retained permanently if desired, or removed after the program has been debugged and is operating satisfactorily. Normal operation of the ASCII/BASIC Module is to run the program in memory immediately after power-up. The run process is normally aborted with a Ctrl/C key sequence, the programmed STOP, or the start-up of ABMHelper.
6. When debugging the overall program, use temporary STOP instructions at key points, the LET instruction to set variables, and the PRINT instruction to view results. Ctrl/C can also be used to halt execution of the program, however it is more difficult to know exactly where the program will be halted.
7. To see the last filenames used during program store and load operations, press the Ctrl/N key sequence while in the terminal emulation mode.
8. If using strings, be sure that the STRING statement is at the very beginning of the program.

Chapter 4 ABMHelper

Features of ABMHelper

The GE Fanuc - NA developed program, ABMHelper is designed to operate with the Series One and Series Five ASCII/BASIC modules. It operates on the same IBM-PC or compatible computer that is normally used for programming the Series One and Series Five PLCs. A summary of features of the ABMHelper program is as follows:

- Provides terminal emulation capability. This provides for basic communication with the ASCII/BASIC Module, and for use of the internal BASIC line editor.
- Provides for archival storage of BASIC programs using the computer's disk storage facilities. Without this capability, your program would only be stored in the ASCII/BASIC Module. This could be a problem, for example, if your ASCII/BASIC Module was somehow damaged, your program would be lost if it were stored only in the ASCII/BASIC Module's memory. With this capability, your program is stored on floppy diskettes which can easily be stored in a safe place.
- Provides a full screen editor useful for creating and editing BASIC programs.

If you have not already read the ABMHelper configuration portion of Chapter 3, it is recommended that you do so now.

System Requirements

The ABMHelper program is designed to run under MS-DOS and a BASIC interpreter, which can be either BASICA or GWBASIC. It has been tested on the following Personal Computer systems, however other IBM-PC compatible systems should work satisfactorily.

Table 4-1. Examples of Systems Tested

Computer	DOS Version	Main Memory	BASIC
IBM-AT	DOS 3.1	640K	BASICA, GWBASIC
EPSON APEX	DOS 3.2	512K	GWBASIC
Workmaster	DOS 3.0	640K	BASICA, GWBASIC
Cimstar I	DOS 3.2	640K	GWBASIC

When running from a floppy disk system, ABMHelper uses a RAM Disk configuration. No additional hardware is required, however, your computer must have at least 384K of main memory.

ABMHelper has been written to use Communications port 1 (COM1). This is the port that normally connects to your ASCII/BASIC Module. Refer to Chapter 3, Installation and Configuration, in this manual for more information on cable connections, or if you want to use a different COM port.

Contents of the ABMHelper Distribution Diskette

The ABMHelper distribution diskette that you received with your system contains the items listed below. These files are found in the main directory of the diskette.

- The ABMHelper program, batch files, and utilities
- The W-ED full screen text editor
- A Demo/Help utility for the W-ED editor
- Text files used by the Demo utility. These files are found in the \TUTORIAL subdirectory on the diskette.

The W-ED editor is useful for many other tasks in addition to being a BASIC editor. You will want to make two copies of the W-ED files, one for use with ABMHelper, and one for use as a general purpose text editor. The procedures for making these diskettes are described in the following paragraphs.

Creating a Backup Distribution Diskette

These steps should always be done.

To make a backup copy of your distribution diskette:

- Put your original diskette in drive A, and the backup diskette in drive B. Do not remove the write protect tab from the distribution diskette. If there is not a write protect tab on the distribution diskette, put one on.
- Use the *DISKCOPY A: B:* command to copy all files on the distribution diskette to the backup diskette. Put the backup diskette away for safekeeping.

Creating a Separate W-ED Editor Diskette

If you want to have a separate Editor diskette for non-ASCII/BASIC Module use, perform the following steps.

- Format another blank diskette. Label this diskette "W-ED Editor".
- Put the original distribution diskette in drive A:, and the new W-ED editor diskette in drive B:.
- Type:

```
A:                  (press the Enter or Return key)
WEDCOPY A: B:       (press the Enter or Return key)
```

This will copy all files associated with the W-ED editor to the W-ED diskette. This diskette can be used for text editing unrelated to the ASCII/BASIC Module or ABMHelper. Also, the operating parameters of the W-ED editor can be changed on this diskette to suit the user's specifications using the INSTALL program, which is discussed later.

NOTE

To use all W-ED features, it is necessary that the W-ED diskette be in the drive that was used to boot up your computer (this also applies to hard disk systems).

GFK-0249

Creating a Workable ABMHelper Diskette (Floppy Disk Based System)

These steps are necessary only if you use ABMHelper on a floppy only (no hard disk) system.

This section describes how to create your ABMHelper working diskette. Instructions for copying ABMHelper to a hard disk can be found later in this chapter.

- Format a blank diskette using the FORMAT/S parameter. Label this diskette "ABMHelper Working Diskette".
- Remove the write protect tab from the original distribution diskette.
- Using the DOS COPY command, copy the following files *from your own computer's DOS diskette* to the original ABMHelper distribution diskette:

```
ASSIGN.COM
VDISK.SYS or other RAM Disk Driver
GWBASIC.EXE or BASICA.COM
```

If your RAM Disk Driver is not named VDISK.SYS, it should be renamed to VDISK.SYS, or the config.sys file must be changed to use the file name of your driver.

- Put the write protect tab back on the distribution diskette.
- Put the original distribution diskette in drive A, and the new ABMHelper working diskette in drive B.
- Type:

```
ABMTOFLP A: B:  (press the Enter or Return key)
```

- This will copy the files needed for ABMHelper to the working diskette. ***The ABMHelper working diskette must be a bootable diskette created with the FORMAT/S parameter.***
- After completing this procedure, you should put your original distribution diskette away for safekeeping.

You now have a working ABMHelper diskette for your computer system. Remember, ***it is necessary to reboot your computer using this diskette (in Drive A) when you want to use ABMHelper from a floppy disk.***

Installing W-ED and ABMHelper on a Hard Disk System

To obtain the full features of the W-ED editor with a hard disk, the W-ED executable files must be located in the root directory of the hard disk. Most features will work properly if the W-ED files are located in a subdirectory. To copy W-ED to the root directory, insert the original distribution diskette in drive A, then type:

```
A:                  (press the Enter or Return key)
WEDCOPY A: C:       (press the Enter or Return key)
```

The above is correct if C is the letter for your hard disk; If not, use the correct letter. This copies all W-ED related files to the root directory of the hard disk. A subdirectory named \TUTORIAL is also created that contains the files necessary to run the W-ED demonstration program.

The following steps are required only if ABMHelper will be used from a hard disk.

- If you will be using ABMHelper on a hard disk system, and the hard disk is drive C, put the ABMHelper diskette in drive A and type:

```
A:                    (press the Enter or Return key)
ABMTOHD A: C:         (press the Enter or Return key)
```

- This procedure creates a subdirectory named \ABM on your hard disk and copy the necessary files to it. The file FDMARK.DAT is not copied to the hard disk since this file is used by the ABMHelper program to determine that it is running in a floppy disk environment rather than a hard disk environment.
- To use ABMHelper, type CD\ABM, then type ABMHD.

If for some reason the file FDMARK.DAT is copied to hard disk, you will notice erratic operation of the program. If this is the case it should be deleted from the hard disk. Also, if you are using the program in a floppy based environment, and if FDMARK.DAT is not on the disk as it should be, you will get a message on entry to the program which says that you are using a hard disk. If this does happen, be sure that FDMARK.DAT is on the boot up floppy. If you use the ABMTOHD batch file as described above, and not a DISKCOPY or COPY command, these problems should not occur.

Using ABMHelper

Invoking the ABMHelper program requires different procedures determined by whether your system has a hard disk or has only floppy disks. To invoke ABMHelper from a hard disk system, the following sequence should be followed:

At the C> prompt, type in the following:

```
CD\ABM                (press the Enter or Return key)
ABMHD                 (press the Enter or Return key)
```

To use ABMHelper from a floppy disk only system is a little more complicated. In order to improve performance, a RAM Disk is created in computer memory, and the temporary files used during BASIC program editing use this RAM Disk instead of a physical drive. It is ABSOLUTELY necessary to reboot your computer with the ABMHelper diskette. By doing this, a special CONFIG.SYS file is read in, and the RAM Disk is created.

Running From Floppy Disk

When properly installed, operation from floppy disk (and RAM Disk) is faster than operation from hard disk. To run the ABMHelper from a floppy disk system, use the following procedure.

- Put the ABMHelper diskette that you created in drive A.
- Press the Ctrl/Alt/Del keys simultaneously, or push the RESET button on your computer.
- After a short period of time you should get a message regarding RAM Disk size and parameters. Note the designation for the drive ID associated with the RAM Disk. Normally, this will be one letter beyond the actual physical drives installed in the system. For example if you have two floppy disks, labeled A and B, your RAM Disk designation will be C.
- Then, type *ABM*. All applicable files will be copied to the RAM Disk. Allowable designations for the RAM Disk drive are B, C, D, E, and F.
- While using ABMHelper, be sure to designate the drive ID when saving or loading files. If you fail to give the drive designator, the file will only be stored in RAM Disk, and may disappear when you exit

ABMHelper. If you accidentally save a program to RAM Disk, and exit ABMHelper, it is still possible to copy the file from the RAM Disk to a real disk using the COPY command, however this must be done before you reboot your computer again, or run other programs.

- Your floppy disks should be in operation ***only when saving or loading a program.*** If your floppy disks operate when using the editor, you have not correctly installed ABMHelper.
- After exiting, if you want to run a different application, it would be best to reboot your computer using your normal boot-up diskette.

ABMHelper Start-Up

The first thing you will see on the screen when you start-up ABMHelper is the following line:

```
1 = S1 ABM,  5 = S5 ABM
```

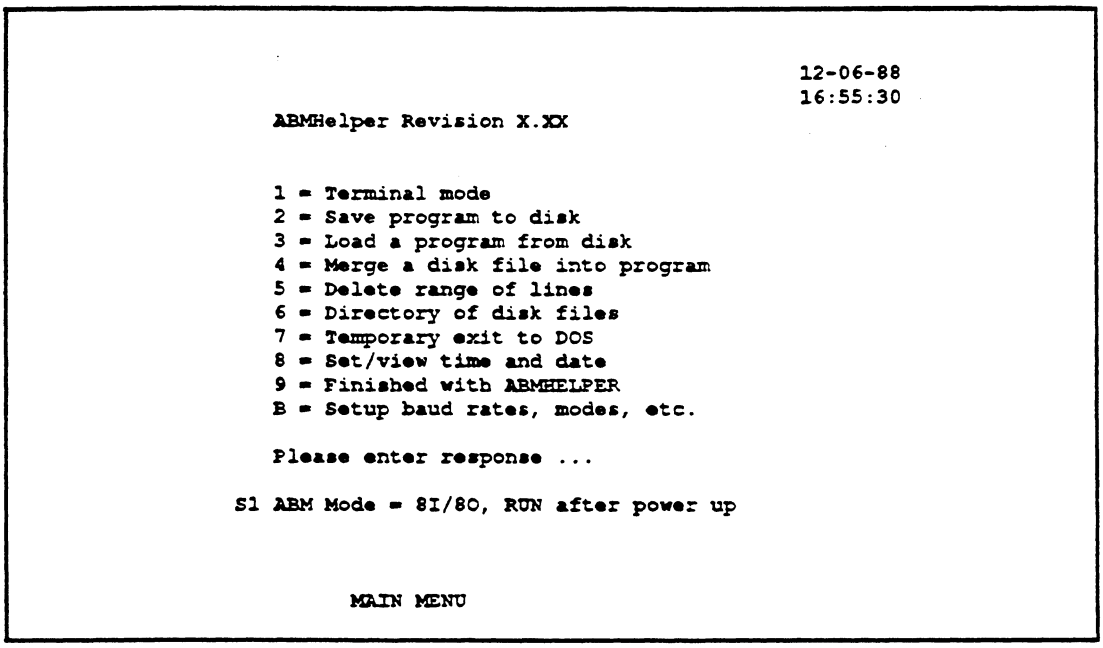
Enter a 1 if you are connected to a Series One ASCII/BASIC Module, or 5 if you are connected to a Series Five ASCII/BASIC Module. *For information on the Series Five ASCII/BASIC Module, refer to GFK-0269, which is the Series Five™ ASCII/BASIC Module User's Manual.*

After entering 1 to select the Series One ASCII/BASIC Module option, one of the following messages will be displayed on the screen.

```
FLOPPY DISK SYSTEM
      or
HARD DISK SYSTEM
```

When ABMHelper starts-up, it will automatically stop the BASIC program execution in the ASCII/BASIC Module. After a short delay the main menu, which provides access to the available functions through selection of a menu option, is displayed. The main menu screen is shown below as it

appears for the "NOSOURCE" mode. Options 2 and 5 will not be used if the "SOURCE" editing mode has been specified.



When the Series One ASCII/BASIC Module is selected, a line will be displayed on the screen indicating the mode that the module is in. The possible modes for the Series One ASCII/BASIC Module and the display for each of those modes are as shown below:

1. CLEAR MODE - This is only used on initial power-up. Type PROG1 when the > prompt is displayed to save the present baud rate. Then power-down and set DIP switch 3 to the ON position.
2. NORMAL OPERATING MODES
   - 8 IN / 8 OUT , RUN after power up
   - 8 IN / 0 OUT , RUN after power up
   - 0 IN / 8 OUT , RUN after power up
   - 4 IN / 4 OUT , RUN after power up

Menu Option 1 - Terminal Mode

Menu option 1 connects the ABMHelper program to the ASCII/BASIC Module for direct input and output from your computer keyboard and screen. When in the Terminal Mode, you can do the following:

- Program the ASCII/BASIC Module.
- Print values.
- Display program output.
- List the program.
- Invoke the full-function editor.
- Set operating parameters.
- Perform other functions, which are described later.

This mode of operation provides most of the features that you will normally use and will be used most of the time.

When this option is selected from the main menu, the following features are enabled:

```
LIST (or list) - Lists (prints) the entire program in ABM memory to the screen.
LIST 100,      - All lines from 100 to the end of program are listed.
LIST 100-250   - Lists the range of lines 100 through 250.
LIST 100-100   - Lists only line 100.
```

Listing part or all of a program fills up the edit buffer with the same information that appears on the screen.

Control functions that have special meaning in this mode are defined in the following table.

NOTE

Do not use the Ctrl/NumLock key sequence to start and stop screen scrolling during a LIST function.

Table 4-2. Interactive/Edit Mode Control Functions

Function	Description
Ctrl/C	Aborts the LISTING in progress, or aborts RUN mode.
Ctrl/S	Stops scrolling on the screen (same as F2).
Ctrl/Q	Starts scrolling (same as F1).
Ctrl/C	Stops execution of the BASIC program (unless the Run/Trap mode has been enabled).
Ctrl/E	Invokes the full-screen editor. When the edited program is reloaded, it merges with the program in memory (memory is not cleared before reloading). The editor works with the information obtained from the last LIST function. *This function is not used when the "SOURCE" editing mode has been specified.*
Ctrl/F	This also invokes the full-screen editor, however, it clears memory before reloading the ASCII/BASIC Module. Using the full-screen editor with Ctrl/F is slower than Ctrl/E since the entire program is uploaded to the edit buffer before editing. It is not necessary to list any of your program before using this function, since the entire program is loaded into the edit buffer. This is the safest method to use until you become familiar with ABMHelper operation. *This function is not used when the "SOURCE" editing mode has been specified.*
Ctrl/P	When in the "NOSOURCE " mode, prints the contents of the EDIT BUFFER. This is normally the last thing that you "listed". If you used Ctrl/F for editing, the entire program will be printed. When in the "SOURCE" mode, prints the file that was last edited. ***Be sure that your printer is "on-line" and connected - otherwise the system will hang-up.***
Ctrl/U	This allows you to call up a user developed external routine from ABMHelper. For example, a user routine could be a directory sort utility. In order to function properly with ABMHelper, the user files must be named as described at the end of this chapter in the discussion on "User Supplied Routines". Use of this feature is recommended only for those users who are very familiar with MS-DOS operation.

Using the F1 - F10 Function Keys

When in the terminal emulation screen, the F1 - F10 function keys are able to perform the functions listed in the following table:

Table 4-3. F1 - F10 Function Keys

Key	Description	Equivalent Control Sequence
F1	Starts scrolling.	Ctrl/Q
F2	Stops scrolling on the screen.	Ctrl/S
F3	Returns to the main menu.	Esc or SUPRV
F4	Returns to the main menu.	Esc or SUPRV
F5	Edits the listed portion of the program.	Ctrl/E
F6	Edits the entire program.	Ctrl/F
F7	Displays the filenames last used.	Ctrl/N
F8	Prints the contents of the edit buffer to the line printer.	Ctrl/P
F9	Aborts the listing in progress.	Ctrl/C
F10	Displays Help information (this page).	Ctrl/I

Menu Option 2 - Save Program to Disk

This option not available in the "SOURCE" editing mode.

Selection of option 2 allows you to store the BASIC program in the ASCII/BASIC Module to your computer's hard disk or floppy diskettes. You are asked for the filename to save to (include the drive ID), and then you are prompted to enter the starting and ending line numbers to be saved. You only need to enter the line numbers if you want to save a part of the program, not the whole program. Be careful not to use the filename of an existing program, unless it is OK to overwrite the file. If you respond with the ENTER (or RETURN) key, to both line number prompts, all of the program is saved.

NOTE

Be aware that any program editing that you have done in the NOSOURCE mode is not saved directly to diskette. To permanently save a program, you must use this option, or use the SOURCE editing mode.

Menu Option 3 - Load a Program from Disk

Selection of this option allows the ASCII/BASIC Module to be loaded from a file stored on a hard disk or floppy diskette. When using this option, the resident program in the program development RAM in the ASCII/BASIC Module is erased before downloading the new program. The drive ID must be specified. Loading a program to the ASCII/BASIC Module takes longer than saving the program to disk. The reason for this is, as each line is transmitted to the ASCII/BASIC Module, it must be converted to executable code by the ASCII/BASIC Module.

Menu Option 4 - Merge a Disk File Into Program

This allows a program stored on diskette to be merged into a program that is resident in the ASCII/BASIC Module RAM memory. The program in the RAM is not erased before the disk file is downloaded. In order to work properly, the program on diskette and the program already in RAM memory must have line numbers that do not overlap. If a line number is read in from the file that has the same line number as an existing line number in the ASCII/BASIC Module, the new line will overwrite the line already in memory. This function is useful if a commonly used routine is to be used in different applications. The new application's line numbers must be designed to not overlap the common routine's line numbers that will be merged later.

Menu Option 5 - Delete a Range of Lines

This option is not available in the "SOURCE" editing mode.

This option allows multiple lines to be deleted from the program in one operation. With menu option 1, a line can be deleted by typing the line number with nothing following it. This is a very slow and tedious procedure if multiple lines are to be deleted. To use this option, you will be prompted to enter the numbers for the start and stop lines (inclusive) to be deleted.

Menu Option 6 - Directory of Disk Files

Selection of this option allows easy access to disk file directories. You will be prompted to enter the file specifications. This specification uses the same format as the DOS directory command. The default is to display all files in the current (RAM Disk) directory. For example, to get a directory display of all

files in drive A with the extension .PRG, you would type *A:*.PRG* in response to the file specification prompt. The /P switch, which causes the directory listing to display only one page at a time is automatically appended to the file specification.

Menu Option 7 - Temporary Exit to DOS

Selection of this menu option allows temporary access to all DOS functions. This option is useful for operations such as deleting or renaming files. To return directly back to ABMHelper, type *EXIT* when you have completed DOS operations. This is an especially useful feature if you are running ABMHelper from floppy disk. You can exit to DOS, run other applications (within limits), and then QUICKLY return to ABMHelper. The normal start-up steps are bypassed.

Menu Option 8 - Set/View Time and Date

Selection of this option provides a direct access to the DOS TIME and DATE functions. It is important to set the correct time and date since this will be the time and date stamped on the files when they are saved to disk.

Menu Option 9 - Finished With ABMHelper

This option is used when you are finished using ABMHelper. You are returned to DOS, and if operating from a floppy diskette, it is suggested that you reboot your computer using your normal boot diskette. If you don't do this, the RAM Disk remains in effect. If you should inadvertently select this option, but have not finished your operations in ABMHelper, immediately restart ABMHelper without rebooting your computer. Any files previously stored in RAM Disk will be retained.

Using User-Supplied Routines with ABMHelper

The ABMHelper program provides you with the capability to easily access prewritten routines. This feature gives you the ability to add custom features to your ABMHelper program. These user written routines are invoked by use of the Ctrl/U character. *Use of this feature by those not experienced in the use of MS-DOS is not recommended.* When a Ctrl/U character is entered from the terminal emulator screen or from the main menu, ABMHelper exits to DOS and executes the file named USER. This file must have an .EXE, .COM, or .BAT extension. The file can be a DOS batch file written by the user, or an executable file that does something useful. If this file uses some other files or DOS commands, it is necessary that these files be present.

To use this feature, you must copy all the required files to the ABMHelper diskette, then rename them. If the file is a DOS batch file, it should have the name *USER.BAT*. If this file is directly executable, it should be named *USER.EXE* or *USER.COM* (do not change the filename extension from the original file specification).

NOTE

> If a directly executable file (not a batch file) is being used, you must provide a way to exit from this file to DOS. This should be a method other than the key sequence Ctrl/Alt/Del, or otherwise resetting your computer.

The user written routines that can be added to ABMHelper can be any useful routine that you wish to use. Some examples of typical routines are described below.

Example 1 - Type a Text File Using Ctrl/U

One routine you may wish to access from ABMHelper is the ability to quickly call-up and print a text file when Ctrl/U is selected from the terminal emulation screen. To do this:

1. Copy the text file to the ABMHELPER working diskette, and rename it *USER1.TXT*.
2. Use the W-ED text editor to create the file *USER.BAT* and enter the following line:

```
TYPE USER1.TXT
```

3. Use ABMHelper as you normally would. However, now when the key sequence Ctrl/U is pressed, the text file will be printed.

Example 2 - Sorted Directory

To have a sorted directory available from ABMHelper, do the following:

1. Create a USER.BAT file by entering the following line:

```
DIR | USER1.EXE
```

2. Next, copy the executable file SORT.EXE to the ABMHelper diskette.
3. Rename SORT.EXE to USER1.EXE.
4. Execute ABMHelper in the normal manner. During start-up, ABMHelper will copy all files that have the form *USER* *.* to the RAM Disk (if using a floppy disk system). To view the sorted directory, enter the Ctrl/U key sequence.

Modifying ABMHelper for use with Fast Computers

When using a computer with a fast operating speed some of the setup parameters may have to be modified to ensure proper operation of the ABMHelper program. The most common problem is noticed when a program is saved from the ASCII/BASIC Module to diskette and the result is nothing in the file, or when a Ctrl E or Ctrl F edit function is selected and the result is a blank screen.

There is a file, named ABM.PSU, included with ABMHelper which provides a means for user modification of ABMHelper. The contents of the file as shipped from the factory are:

```
9600
COM1
LPT1
NOTIMEDATE
200
200
600
400
400
NOSOURCE
```

The first 200 in the file is the item that typically needs to be modified, depending on the computer. The value 200 works properly on an AT computer or a Workmaster computer. On a faster computer, this value must be at least 800. This file can be modified using the W-ED editor provided with ABMHelper, or with EDT or almost any other editor that creates an ASCII file. Care must be taken not to change any of the other parameters in the file.

If the value is too large, such as 800 for a Workmaster computer, after the ASCII/BASIC Module status LEDs stop blinking (no more data being received) the Workmaster computer will be idle for a long time before the editor screen appears.

Troubleshooting ABMHelper

The ABMHelper program provides an easy-to-use enhancement to the Series One ASCII/BASIC Module. However, if you encounter any problems trying to operate it for the first time, or after you have been using it, there are some basic troubleshooting procedures to follow that should quickly solve your problem. Following is a list of problems that you may encounter and the probable solutions to solve these problems.

1. You cannot get the system running initially.
   - Refer to the installation instructions, and carefully repeat the installation procedure.
   - Ensure that all commands and other entries are entered correctly.
2. ABMHelper seems to be working properly, however you get a message indicating that ABMHelper and the ASCII/BASIC Module are not communicating.
   - Check all cable connections.
   - Ensure that the ABM.PSU file has not been deleted from the disk.
3. A message is displayed stating that ABMHelper will not work with the version of ASCII/BASIC Module that it is connected to.
   - Consult your GE Fanuc - NA Distributor, GE Fanuc - NA Sales Office, or the factory.
4. Your system has been working properly, but does not now appear to be doing so.
   - Enter the key sequence *Ctrl/C* to abort operation.
   - Enter the key sequence *Ctrl/Q* to resume scrolling.
   - If a printer is connected, and you have printed a program, be sure that the printer is on-line.
   - Enter the key sequence *Ctrl/Break* (on an IBM ®-PC or compatible keyboard), or *Ctrl/BIN* on a Workmaster computer keyboard. Then Type *RUN*, and press the ENTER (or RETURN) key.
   - If none of the above steps work, press the key sequence *Ctrl/Alt/Del,* to reset the computer.
   - You can also reset your computer by pushing the reset button (if the computer has one), or by cycling power to the computer.
5. You are using a floppy disk system; you get the message *HARD DISK SYSTEM*, and every time that you use the LIST command, you notice that the floppy disks turn-on.
   - It is possible that your FDMARK.DAT file has been accidently deleted from your ABMHelper floppy diskette.
   - From DOS, use the W-ED full screen editor to create the file, FDMARK.DAT.
   - This file does not need to have any data in it.
   - When exiting the W-ED editor, use the F1/RETURN key sequence.
6. You are using a hard disk system, and you get the message *FLOPPY DISK SYSTEM.*
   - Delete the file FDMARK.DAT from the hard disk.
7. You cannot use ABMHelper in an off-line mode without an ASCII/BASIC Module physically connected. The system will hang up when you enter the terminal emulation mode.

- If this happens, you need to either connect an ASCII/BASIC Module, or you may need to reboot your computer.
- It is possible, however, to use the W-ED editor, or any other editor that creates ASCII text files to create an off-line BASIC program, then download the program with ABMHelper after communications are established.

General Hints

- Do not use READY or LIST in any remarks.
- Be careful not to use keywords in variable names. For example, the following statement is illegal since LIST is embedded in the variable name:

```
LET QULIST5=10
```

- If your system does seem to hang up, you can usually recover by using the Ctrl/C key sequence.
- It is also possible that you may have inadvertently entered a Ctrl/S (Stop Scrolling) character. Try pressing the Ctrl/Q keys to begin operation again.

Chapter 5
The W-ED Editor

GFK-0249

The W-ED editor portion of this manual was written assuming the use of an IBM®-PC type keyboard. When using the Workmaster® computer, toggle the computer's keyboard to the 83-key mode by pressing the Ctrl/Select keys. The Workmaster computer's keys will then function as described below.

Table 5-1. PC Keyboard Conversion Chart

Workmaster Computer Keyboard	Functional Description	IBM PC-Type Keyboard
ABORT	Insert line below	F9
HELP	HELP	F10
SUPR	Esc	Esc
F/9	Page Up	PgUp
^F/9	To top of document	^PgUp
3/Decr	Page Down	PgDn
^3/Decr	To bottom of document	^PgDn
7/D	To start of line	Home
^7/D	To start of page	^Home
1/Incr	To end of line	End
^1/Incr	To end of page	^End

Editor Command Summary

A summary of the editor commands useful during BASIC programming and the keystrokes required to use them is provided below. These commands can be selected after Ctrl/E or Ctrl/F has invoked the W-ED editor in NOSOURCE mode, or after the W-ED editor has been called in SOURCE mode. Note that additional commands are useful when using W-ED in a general purpose text editing environment.

```
F1 - Exit                    Alt L - mark line       ^PgUp - Top of file
F2 - Execute Command line    Alt U - unmark block     PgUp - Top of screen
F3 - Enter/Insert line       Alt C - Copy block       PgDn - Down 1 screen
F4 - Quit                    Alt M - Move block      ^PgDn - Bottom of file
F5 - Delete line             Alt D - Delete block    ^Home - Start of screen
F6 - Erase to End Of Line    Alt E - Erase to EOL     Home - Start of line
F7 - Split line                                       End  - End of line
F8 - Display tabs                                    ^End  - End of screen
F9 - Insert line below       COMMAND LINE - Esc, then:
F10- Help                    /str/  - forward find,  -/str/  - backward find
```

Changing W-ED's Configuration

To modify the operating parameters of the W-ED editor, type *INSTALL*, then press the ENTER key and follow the on-screen instructions (On-line HELP is available (F10) for each item). You may want to do

this when using W-ED as a text editor for programming other than with the ASCII/BASIC Module. The default settings for using W-ED as a BASIC editor are listed below.

Table 5-2. Default Settings for W-ED Editor

Default Setting	Description
Cursor placement †	Top of text area
Color definition †	07H
Input formatting ‡	Word wrapping; no reformat
Enter key toggle †	ON
Border option ‡	ON
Screen mapping ‡	Through ROM BIOS
Line feed char ‡	YES
User paging char †	007
Default lines per page †	55

† Can be changed to suit user's taste.

‡ Change not recommended for ASCII/BASIC Module programming.

A Quick Introduction to W-ED

The following section provides you with a quick and easy introduction to using the W-ED editor. Experienced computer users will be able to use W-ED effectively in just a few minutes because it uses common conventions and practices.

For a more detailed discussion of using W-ED, run the demo program provided on the distribution diskette. This quick introduction is not complete. However, it will get you started so that you will be in a position to "know what you don't know" in order to use the tutorial material or the demo program for reference. To run the demo program, type:

```
DEMO                (press the Enter or Return key)
```

then follow the instructions on the screen.

To help you use the quick introduction, the PgDn key scrolls the file down one screen at a time. PgUp does the opposite. You can also scroll using the cursor-up and cursor-down keys.

To get a hard copy of the W-ED tutorial plus miscellaneous information, type *MAKEMANL* at the DOS prompt before running ABMHelper. This creates a file named MANUAL, which can then be printed.

Starting W-ED From DOS

W-ED is activated by typing:

```
WED filename.ext
```

at the DOS prompt. If no filename is used, W-ED assumes a new file. You will be able to assign the filename of your choice when you leave W-ED. Paths are supported by W-ED.

GFK-0249

W-ED Function Keys

After entry to W-ED, the actions caused by pressing the other function keys are shown below.

Write file to disk	F1	F2	Do task on command line (Search)
Change (Enter) key action	F3	F4	Quit - abandon W-ED to DOS
Delete entire line	F5	F6	Erase to end of line
Split line _ at cursor	F7	F8	Set margins (L and R) and tabs (T)
Insert line below cursor	F9	F10	Help - browse for more information

Other W-ED Key Actions

- The Esc key moves the cursor between the TEXT AREA and the COMMAND line.
- The Esc key also abandons (aborts) operations, such as the Help screen.
- W-ED uses most IBM-PC keys normally. The cursor keys work normally. The cursor will wrap around line ends and cause the screen to scroll.
- Some actions are invoked by using Alt key combinations. For example, to select Alt I, hold down the Alt key and press the I key.
- Some keys, when combined with the Ctrl key (written as ^), have extended powers.
  - ^Tab moves the cursor right one word. ^Shift Tab moves the cursor left one word.
  - Home moves the cursor to the left margin.
  - ^Home moves the cursor to the top left of the screen.
  - End moves the cursor one position beyond the last character on a line.
  - ^End moves the cursor to the bottom left of the screen.
  - ^PgUp displays the beginning of the file on the screen.
  - ^PgDn displays the end of the file on the screen.

W-ED Features

- You will have an opportunity to change the file specification before you actually save a file. Simply edit the filename using W-ED conventions.
- Invalid file specifications will not be accepted, and you will have an opportunity to correct the specification.

Command Line and Text Area

There is a clear distinction between Command line functions and text area functions as listed below.

- Press the Esc key to move the cursor to the TEXT area.
- Press the Esc key to move the cursor to the COMMAND LINE.
- When entered on the COMMAND LINE */string/* will locate the "string" after the F2 key is pressed.
- Forward and backward string searches are case insensitive.

Block or Cut and Paste Operations

There are ten block operations available with W-ED.

Table 5-3. Block Operations

Operation	Description
ALT L (letter L)	Defines the (first) line of a block marked with the cursor. Moving the cursor and typing ALT L marks the end of the block, if it is more than one line.
Alt D	Deletes a block.
Alt C	Copies a block starting on the line below the cursor.
Alt M	Moves a block starting on the line below the cursor.
Alt V*	Centers a block (usually a line).
Alt R*	Reformats a block of text into paragraph form.
Alt J*	Left and right justifies text between the margins.
Alt T	Capitalizes all letters in a block.
Alt S	Makes all letters in a block lower case.

* These commands should not be used when editing BASIC programs, however they are useful in other situations.

Tabbing and Margins

Press F8 then press the Tab key several times.

- L and R mark the margins, and T marks the tab stops.
- Changing the line and pressing Enter, sets new margins and tabs.
- Back tabbing (Shift Tab) is supported.
- Tabs are written to disk as blanks.
- Pressing the Tab key does not move text to the right, it only moves the cursor.

Extended ASCII (Graphics) Characters

- W-Ed normally allows only characters that are visible on the keyboard to be typed.
- Alt G enables use of extended or graphics characters. A "G" appears on the Command LINE when this selection is made.
- Extended characters are entered using the Alt key and the number pad.

Printer Paging Control

- The page length can be set using ^p.
- Page breaks can be set automatically using ^p or manually using Alt P.
- Automatic page breaks coordinate with manually set breaks.
- Page breaks are visible on the screen.
- Manual page breaks can be deleted just like any other line of text.
- Upon re-edit, automatic page breaks are removed by W-ED.

Text Manipulation

- Word wrap can be turned on and off using Alt W.
- Word wrap status is displayed on the COMMAND LINE.
- The Enter key can be set to insert a new line or to move the cursor down one line and to the left margin using F3 or Alt X.
- The Enter key status is displayed by an X on the COMMAND LINE.
- The Backspace key moves the cursor to the left while deleting characters.
- The Del (Delete) key keeps the cursor in the same position but deletes characters to the right of the cursor.
- Both the Del and the Backspace keys close-up the line by moving the rest of the line to the left.
- Inserting characters and words can be done in two modes: type-over (overstrike) mode and open-the-line (insert) mode.
- Insert mode status is displayed by an I on the COMMAND LINE.
- Insertions causing word wrap "open" the file to allow easy further manipulation (words down the left margin).
- Partial lines may be deleted to the left or right of the cursor in one or at most two key strokes. F6 deletes the line to the right of the cursor, and splitting the line using F7 and the line delete key F5 will delete the left side of the line.
- Blank lines may be inserted above the cursor position using Alt A.
- Blank lines may be inserted below the cursor position using F9.
- Lines may be split onto two lines by marking the position where they are to be split with the cursor and pressing F7. When using W-ED as a general purpose editor, it is often necessary to perform a BLOCK operation, such as DELETE or REFORMAT, however you want the operation to affect only part of a line. To accomplish this the F7 function breaks the line into two lines (one below the other) at the cursor location, where the block functions can then be made to perform as desired.
- To insert a file in another file while using W-ED, use the following procedure:
  - Start editing the file that is to have the other file inserted (call this FILE1).
  - Press the Esc key to go to the command line, then type "WED FILE2". This will put you in FILE2.
  - Using Alt Ls, mark the beginning and end of the block to be inserted in FILE1.
  - Press the F4, Return, and Esc keys. This will return you to FILE1.
  - Move the cursor to the location where you want FILE1 to be inserted, and press Esc, Ctrl/A. The marked part of FILE2 will be inserted at the cursor location.
- Global search and replace can be done using the W-ED editor by following these steps:

- Be sure W-ED is in the appropriate (usually overstrike) mode.
- Press the Esc key and enter the string to be replaced on the command line, e.g. "/ old /". If you don't want an embedded "old" to be replaced, e.g. in the word "bold", be sure to include spaces before and/or after the string to be replaced as shown above.
- Type Alt Y. You will see a blinking Y on the command line indicating that the next sequence of keys will be repeated.
- Press the F2 key. The cursor will move to the first occurrence of "old". Type the new information over the old, e.g. "new".
- Then press Ctrl/Y, Ctrl/Y. All occurrences of "old" will be replaced by "new".
- To abort the replace operation, press the Esc key.

• When using the W-ED editor for general purpose use, a messy document is often temporarily created during the process as a side effect of W-ED operation. To clean-up the messy area, mark the beginning and end of the area with Alt L, then use Alt R or Alt J to reformat the area, and Alt U to "unmark" the area.

• When using the W-ED editor for general use, it is often necessary to indent a paragraph or other section of the document. This can be accomplished by resetting the L tab (F8) to the required indent position, then use Alt L, Alt R and Alt J keys as described above to reformat that area. *Don't forget to set the L tab back to its original location. Also, do not use the Alt R or Alt J reformat commands when editing an ASCII/BASIC Module file. If this is done accidently, you should quit by using the F4 key, then start the edit session over again.*

The information presented above should be sufficient to get you started. With a little practice, you will soon master using W-ED.

W-ED Messages

W-ED messages are listed below alphabetically. Each message is followed by an explanation and a suggested action. Messages which appear when exiting W-ED are marked with an asterisk.

ACCESS DENIED. CHECK SPECIFICATION. *

You attempted to edit a file marked as read only. To edit the file, the file attribute in the DOS Disk Directory must be modified. You attempted to save a file to diskette without giving it a filename.

ARE YOU SURE? (Y/N)

You made changes to the file and pressed the F4 key to QUIT. The changes made to the file will be lost and the file will remain as it was before you edited it. Decide if this is the action you intended; if not, press the Esc key cancelling the QUIT command.

AT END-OF-FILE !!!!

In non-word wrap mode, if you attempt to type past the margin on the last line, this message will appear. On lines before the last line, the cursor will wrap to the beginning of the next line. You can create a new line by using the F9 key.

BAD TARGET LOCATION

You attempted to move or copy a block or line into itself. Move the cursor to another location, or redefine the block of text to be manipulated.

BUFFER IS FULL.....

The buffer in W-ED holding the file being edited is full. No more text can be added to the file. Save the file using the function key F1, and then re-enter W-ED with the same filename. Find a convenient or logical place to divide the file into two parts. Delete one part and save the file under a new filename. Re-enter W-ED again with the original file - the one that was too large. Delete the other part of the file and save the file under a second new name. Now each part of the original file can be edited further.

The DOS copy command will let you recombine the two parts for printing or other purposes. For example, if FILETOO.BIG fills the buffer and you divided it into FILETOO.ONE and FILETOO.TWO:

```
COPY FILETOO.ONE+FILETOO.TWO BIGFILE.NEW
```

BYTES FROM DISK ------ NO EOF: nnn *

BYTES OUT OF EDITOR -- NO EOF: nnn

BYTES TO DISK -------- INCLUDING EOF: nnn

The above message is displayed for information only when a file is saved to disk. "BYTES FROM DISK" is the decimal number of bytes read into W-ED from disk, EXCLUDING the end of file marker(s).

"BYTES OUT OF EDITOR" is the decimal number of bytes returned from W-ED, again, EXCLUDING the end of file marker(s).

"BYTES TO DISK" is the decimal number of bytes actually written to disk from W-ED. This includes the end of file marker(s). This number should always be a multiple of 128.

CANNOT ACCESS DISK X ...

Disk drive X is not available. Save the information on an available drive.

CANNOT ALTER PAGING

You attempted to modify an AUTO or USER PAGING line. Deleting the line (e.g. F5) is the only alteration allowed. All AUTO PAGING may be deleted by typing Ctrl/P, Ctrl/D.

CMND NOT RECOGNIZED

Only searching and moving forward or backward "nn" lines are valid in command line operations. The search command starts with "/" or "-/" and is followed by a string of characters to locate. Moving forward or backward "nn" lines is commanded by typing a number or a minus sign and a number respectively.

DISK IS WRITE PROTECTED *

You attempted to save a file to a write protected diskette. Change the file specification to specify another disk drive or replace the disk with another one which is not write protected.

ENTER ESC (^D=DEL)

After pressing ^P to define auto paging, this message appears defining the options available in this mode. ENTER will accept the number of print lines/page specified, and cause the AUTO PAGING to occur. Esc will allow you to escape from the auto paging mode. ^D will remove all auto paging lines previously entered into the file.

ENTER IF SAME -- OTHERWISE MODIFY AS NEEDED *

When the F1 (FILE) function key is pressed you may save the file being edited under a new name, or define a new path. If you wish to save the file under a new name or a new path, type the new filename over the old one, and press ENTER. If no change is desired, press ENTER.

FIELD IS IN MARGIN

While searching for a string of characters using the "/" command on the command line, the target string was found; however, the string starts to the left of the left margin or to the right of the right margin. Since the cursor cannot travel outside the margins set by the tab command, W-ED puts the cursor at the margin and prints this message on the message line.

When margins are reset during editing, the message is displayed if the cursor is outside the area defined by the new margins. Moving the cursor causes the message to disappear.

FILE SPECIFICATION ACCEPTED..... *

The filename or path of the file you specified to save are acceptable to DOS.

FILE TOO LARGE TO LOAD INTO EDITOR

W-ED detected that the file you have named to edit will not fit into W-EDs internal buffer. W-ED cannot be used to edit the file.

FN: 1F 4Q 5D 9I 10H

Normal message displayed. The abbreviations are :

```
 1F ==> F1  = FILE (Save changes)
 4Q ==> F4  = QUIT (Don't save changes)
 5D ==> F5  = DELETE LINE
 9I ==> F9  = INSERT LINE
10H ==> F10 = HELP
```

ILLEGAL DRIVE SPECIFIED *

Valid drive specifications are A and B if you have 1 or 2 drives attached; A, B, and C if you have 3 drives attached, etc. Attempting to access a drive, e.g. drive F, which does not exist in your system will cause this message to appear.

INSUFFICIENT BUFFER SPACE TO PROCESS FILE

The file you attempted to load into W-ED is too big for W-ED to handle. W-ED will handle a maximum of 750 lines of 78 characters.

GFK-0249

INVALID DRIVE OR NOT ENOUGH ROOM FOR FILE *

When DOS attempts to write a file to disk, if the drive is not specified correctly, or if there is insufficient room on the diskette to hold the file to be written to disk, DOS returns this error message. In W-ED, it is most likely that there is insufficient room on the disk, as an illegal drive specification would normally be detected earlier.

LINE ALREADY FULL

You attempted to insert characters in a line, and word wrap is off. When word wrap is on, the last word on the line will be moved to a new line just below the current line. When word wrap is off, the last word on the line will not be moved to a new line. Delete a character, press Alt W to turn on word wrap, and then continue inserting characters.

MAX 15 TABS ALLOWED

You specified more than 15 tab stops, including margins. W-ED will handle no more than 15.

NO BLOCK DEFINED

Before you can use the block commands such as copy, delete, or move a block of text, you must define it using Alt L. Mark the block of text to be operated on with Alt L marking the beginning and ending lines. Then use the block command such as copy, delete, or move.

NOT ALLOWED HERE

When the cursor is on the command line, the block commands and Alt A and Alt E cannot be used since they require the cursor to be positioned in the text area at the action site. Move the cursor to the desired position, and execute the command again.

NOT FOUND

The string of characters specified in a search command was not found in the file.

OLD BLOCK STILL SET

A block is already defined somewhere. Unblock it with Alt U, and then define a new block.

SPECIFIED PATH NOT FOUND *

You apparently altered the path or name during an F1 (FILE) operation and the specified path is not valid. Save the file using another path, then check if the subdirectory exists and the path is correct.

TABS:(ESC) ENTER

Indicates that the tab mode has been entered by pressing F8.

TOO MANY OPEN FILES

If W-ED is called from another program, it is possible that too many files have already been opened, and the new file which is to be read by W-ED is above the maximum.

TRY SMALLER BLOCK

You made a block (using Alt L commands) too large for W-ED to work with. W-ED reserves some of the file buffer space for uses such as moving blocks or text. As a file grows, this area diminishes. Use Alt U to unblock the block you defined, and perform the block activity in several steps.

....NEW FILE....

A file of the specification name in the given path was not found. W-ED assumes that a new file is being created and displays this message. If you are trying to edit a file you created previously, check the path to the file and make sure you have the correct diskettes in the drive.

??? MARGINS IGNORED

When setting margins, the leftmost tab must be an "L" and the rightmost tab must be an "R". If this is not the case, or if you have specified multiple Ls and Rs, you will get this message.

(Cursor-up symbol) OR (Cursor-down symbol) --- ESC TO EXIT

From inside the HELP panel, pressing the up or down arrow keys scrolls the HELP screen; pressing Esc returns you to the edit mode.

GFK-0249

Command: RUN(cr)

Action Taken:

After RUN(cr) is typed, all variables are set equal to zero, all BASIC-evoked interrupts are cleared, and program execution begins with the first line number of the selected program. The RUN command and the GOTO statement are the only way you can place the MCS BASIC-52 interpreter in Run mode from the Command mode. Program execution may be terminated at any time by typing Ctrl/C on the keyboard.

Variation:

Unlike some BASIC interpreters that allow a line number to follow the RUN command (i.e., RUN 100), MCS BASIC-52 does not permit such a variation on the RUN command. Execution always begins with the first line number. To obtain the same functionality as the RUN(ln num) command, use the GOTO(ln num) statement in Command mode.

Example:

```
>10 FOR I=1 TO 3
>20 PRINT I
>30 NEXT I
>RUN

 1
 2
 3

READY
>
```

Command: CONT(cr)

Action Taken:

If a program is stopped by typing Ctrl/C on the keyboard or by executing a STOP statement, you can resume execution of the program by typing CONT(cr). Between stopping and restarting the program, you may display the values of variables or change the values of variables. However, you may *not* CONTinue if the program is modified during the STOP, or after an error.

Variation::

None.

Example:

```
>10 FOR I=1 TO 10000
>20 PRINT I
>30 NEXT I
>RUN

1
2
3
4
5    - (TYPE Ctrl C ON KEYBOARD)

STOP - IN LINE 20

READY
>PRINT I
 6

>I=10

>CONT

 10
 11
 12
```

Command: LIST(cr)

Action Taken:

The LIST(cr) command prints the program to the console device. Note that the LIST command "formats" the program in an easy-to-read manner. Spaces are inserted after the line number, and before and after the statement. This feature is designed to aid in debugging MCS BASIC-52 programs. The "listing" of a program may be terminated at any time by typing Ctrl/C on the keyboard.

Variation:

Two variations of the LIST command are possible with MCS BASIC-52. They are:

```
LIST [ln num](cr)

and

LIST [ln num] - [ln num](cr)
```

The first variation causes the program to be printed from the designated line number (integer) to the end of the program. The second variation causes the program to be printed from the first line number (integer) to the second line number (integer).

NOTE

The two line numbers *must* be separated by a dash (-).

Example:

```
READY
>LIST
 10 PRINT "LOOP PROGRAM"
 20 FOR I=1 TO 3
 30 PRINT I
 40 NEXT I
 50 END

READY
>LIST 30-30
 30 PRINT I

READY
>LIST 20-40
 20 FOR I=1 TO 3
 30 PRINT I
 40 NEXT I
```

Command: NEW(cr)

Action Taken:

When NEW(cr) is entered, MCS BASIC-52 deletes the program that is currently stored in RAM memory. In addition, all variables are set equal to zero, and all strings and BASIC-evoked interrupts are cleared. The real time clock, string allocation, and internal stack pointer value (location 3EH) are *not* affected. In general, NEW(cr) is used simply to erase a program and all variables. However, this command does *not* erase the contents of the archival storage area or the permanent storage area.

Command: NULL [integer](cr)

Action Taken:

The NULL [integer](cr) command determines how many NULL characters (00H) MCS BASIC-52 will output after a carriage return. After initialization, NULL = 0. The NULL command was needed when a "pure" mechanical printer was the most common I/O device. Most modern printers today, however, contain some kind of RAM buffer that virtually eliminates the need to output NULL characters after a carriage return.

The NULL count used by MCS BASIC-52 is stored in internal RAM location 21 (15H). The NULL value can be changed dynamically in a program by using a DBY(21)=[expr] statement. The [expr] can be any value between 0 and 255 (0FFH), inclusive.

Variation:

None.

Description of GE Fanuc - NA Custom Commands

Several custom commands have been developed by GE Fanuc - NA especially for use with the Series One ASCII/BASIC Module. These commands can also be used with the Series Five ASCII/BASIC Module. The custom commands add programming capability in addition to the Intel MCS BASIC-52

programming language built into the ASCII/BASIC Modules. The custom commands are described below.

Transfer Command: TRANSFER R/W, FORMAT, ABMLOC, FLAG

Table 6-1. R/W Parameter Description

Value	Description
R	Read data from the Series Five CPU.
W	Write data to the Series Five CPU.

A non R, W or incorrect format for I/O setup will result in the error message *WRONG OR MISSING PARAMETER IN LINE XXXX.*

NOTE

Only the 8 input/0 output mode is invalid for W (Write), all other modes are valid for Write.
Only the 8 output/0 input mode is invalid for R (Read), all other modes are valid for Read.

If FORMAT = 2, the data in the Series One CPU is in BINARY format.

If FORMAT = 4, the data in the Series One CPU is in BCD format.

Data Format and Range

Valid limits for data range for the different addressing modes are listed below. These reference ranges must be followed for proper operation of your program.

Table 6-2. I/O Reference Ranges

FORMAT	8 Out (R)	8 In (W)	4 Out (R)	4 In (W)
2, BIN	0-255	0-255	0-15	0-15
4, BCD	0-99	0-99	0-9	0-9

For 8 Out (R), 4 Out (R) the data from the Series One CPU is checked to ensure that it complies with the valid ranges.

For 8 Input (W), 4 Input (W) the data in the variable to be sent is checked.

ABMLOC

A scalar or array variable name used in the program, will contain data to/from the Series One CPU. ABMLOC should be defined before the execution of the Transfer command. If the Transfer command is executed without ABMLOC being previously defined, FLAG will be set to 2. If a statement such as ABMLOC=0 is later executed, it becomes defined at that time. On subsequent transfer attempts, ABMLOC will then be defined, so Flag will not equal 2 from that point on. To define it earlier in the program, a statement such as X=0 can be used, where X is the ABMLOC parameter.

FLAG

A scalar or array variable name used in the program, which shows the status of the transfer command. The values for the FLAG command, and the associated transfer status are:

Flag Value	Status
-1	Transfer is not completed
0	Transfer has been completed successfully
2	ABMLOC not yet defined
3	Range error in the data to be transferred
None of the above	Error condition

Example of Using Flag

The following programming example shows how Flag can be used in a program.

```
1000  REM --- FLAG USAGE EXAMPLE ---
1010  REM
1020  LET I = 0
1030  LET FLAG = 99
1040  TRANSFER W,2,I,FLAG
1050  IF FLAG = -1 GOTO 1050 : REM TRANSFER NOT COMPLETE
1060  IF FLAG <> 0 GOTO 2000 : REM ERROR
1070  REM --- TRANSFER COMPLETE ---
1080  REM --- CONTINUE WITH REMAINDER OF PROGRAM ---
 .     .
 .     .
 .     .
2000  REM --- ERROR ROUTINE ---
2010
```

Example of Timeout to Detect CPU in STOP Mode

The following program example sets a value, which allows the program to detect when the CPU is in the STOP mode.

```
1000  LET X = 0
1010  LET FLAG = 99
1015  CLOCK 1
1020  LET DBY (71) = 0 : REM SET FRACTIONAL PORTION OF TIME TO 0
1025  LET TIME = 0
1030  TRANSFER R,2,X,FLAG
1040  IF TIME >1.0 THEN GOTO 2000
1050  IF FLAG <> 0 THEN GOTO 1040
1060  GOTO 1020
2000  PRINT "TIMEOUT"
2010  GO TO 1020
```

Functions Used in RUN Mode

All of the following functions can be used in mathematical expressions when writing a BASIC program.

LSTR

This function returns the length of a string, i.e., the number of characters contained in the string.

```
LSTR $(n),var
```

Where n = the number of the string variable, and var = the variable where the length of the string will be stored.

SPLIT

This function is used to read a single bit or a field of bits.

```
SPLIT [var1],[var2],S,L
```

Where var1 is the variable to be examined; var2 is the target variable; S is the starting bit (valid range 1 to 16); L is the number of consecutive bits to be addressed moving left to right from the starting bit. An example of this function and the results of using it are shown below.

```
SPLIT [var1],[var2],9,3

 bit  16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
data   x  x  x  x  x  x  x  1  1  0  x  x  x  x  x  x
                           |_____|
  var1 integer bit pattern


 bit  16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
data   0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0
                                                |_____|
                  var2 integer bit pattern

                                          x = don't care
```

MOVE

This function is used to set a single bit or a field of bits.

```
MOVE [var2],[var1],S,L
```

Where var1 is the target variable where the bit or bits will be set; var2 is the source variable; S is the starting bit, with a valid range of 1 to 16; and L is the number of consecutive bits to be addressed moving left to right from the starting bit. An example of using this function is shown below.

```
MOVE [var2],[var1],12,4

 bit  16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
data   0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  1
                                          |_________|
                  var2 integer bit pattern


 bit  16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
data   x  x  x  x  1  0  1  1  x  x  x  x  x  x  x  x
                  |_________|
   x = don't care             var1 integer bit pattern
```

```
Error condition:

     * Starting bit >16 or <1
     * The value of the length parameter exceeds the value of the start
       address, (S < L).
```

SWITCH

This function, when executed, returns the value of the setting of the DIP switch on the ASCII/BASIC Module.

```
SWITCH [var]
```

MATCH

This is a functional operator which will set a variable flag with the value -1 (true) or 0 (false) as a result of a string comparison.

The form of the command as shown below is used to test the content of the string to match the data format specified by the mask shown in brackets:

```
MATCH [string],(99AAA999XX),flag
```

Where 9 can be any number from 1 to 9; A can be any letter a to z or A to Z; X can be any valid ASCII character; flag is the variable that will contain the result of the comparison. An example of using this function is given below.

```
10 INPUT $(1)
20 MATCH $(1),(a...z , A...Z),f1
30 IF f1=0 THEN 10 ELSE 40
40 .......
```

This function will also test whether two strings are identical. The form of this function is shown below, followed by an example of its use.

```
MATCH [string],"STRING",flag

MATCH [string],$(n),flag          n = number of the string
```

```
10 INPUT $(1)
20 MATCH $(1),"yes",f1
30 IF f1 THEN 100
40 MATCH $(1),"no",f12
50 IF f12 THEN 200 ELSE 10
.....
```

INPB

The INPUT statement is normally used to input data to the program from a keyboard. This data is normally ASCII coded (refer to the ASCII code chart in Appendix E). When using INPUT, a 03H coming in the port will be interpreted as a Control C,and will abort processing. Similarly, 13H will be interpreted as a Control S and inhibit output from the module. This is normally desirable for keyboard processing.

For some applications, however it is necessary to read all the data, store it away, and not perform any actions upon the data, except as specified by the BASIC program itself. A common example is a communications protocol which uses binary (not ASCII coded hex) data.

The INPB instruction was developed to meet this need. This instruction gets a specified number of characters from the serial port, and stores them in the specified string. If the timeout value is reached, and the specified number of characters has not been fetched, the instruction sets ***flag*** to an error value, and continues with the next BASIC instruction.

The format for the INPB instruction is as shown below.

```
INPB $(n), number of characters, timeout value, flag
```

where:

```
$(n) - array to store characters - STRING must have already
       allocated space.

number of characters - 1 to length $(n) - must be a constant, and
       cannot exceed the length of $(n).

timeout value - 1 to 65535 ms.

flag = variable - 0 = no error
                  1 = timeout
                  2 = Number of characters larger than $(n)
                  3 = Number of characters <1, or timeout <0, or
                      timeout >65535, or number of characters
                      not defined as a constant.
```

It should be noted that the string is not cleared before the INPB instruction writes to it. If, for example, an INPB instruction writes ten characters to string $(1), and another INPB instruction writes two characters to string $(1), then string $(1) will consist of the first two characters from the second INPB, and the last eight characters from the first INPB. Also, FLAG must be declared prior to executing this instruction.

NOTE

While INPB is waiting for characters, the module ***will not*** respond to Ctrl C (Abort). For a program using INPB, it may be necessary to enter the Ctrl C sequence several times to Abort the program.

Commands Used in Command (Monitor) Mode

RENUM@

This command is used to renumber all program lines from the specified starting line to the last line in the program, and all references to these lines in the program. This function allows you to enter new

program lines where previously there had been no room for these lines. The renumbered program can start with any line number, and affect only part of the old program. The valid formats for this command are as shown below.

```
RENUM@
RENUM@ Newline
RENUM@ Newline, Startline
RENUM@ Newline, Startline [increment]
```

Where *Newline* is the first line number to appear in the new (renumbered) program, *Startline* is the first line number in the old program which will be affected by the RENUM@ command (all subsequent lines will also be affected), and *increment* is the difference between successive lines in the new (renumbered) program..

Error conditions for this command are:

```
* Startline does not exist
* INCR is < 1
* Line number generated is >65535
```

The default conditions for this command are:

```
Newline = 10
Startline = first line of program
Increment = 10
```

All instructions that branch to lines that are renumbered (GOTO, GOSUB, ON GOTO, ON GOSUB, ONERR, ONTIME) are automatically changed to conform with the new line number. An example of using this command is shown below.

```
Original Numbering        Command             New Numbering

    10 A=1          --- RENUM@ 10,10,10 --->     10 A=1
    20 B=2                                       20 B=2
    30 GOTO 101                                  30 GOTO 110
    .                                            .
    .                                            .
    .                                            .
    90 C=3                                       90 C=3
   100 D=4                                      100 D=4
   101 E=5                                      110 E=5
   102 F=6                                      120 F=6
```

DELPRM

This instruction deletes a program stored in the Archival Storage Area that had been saved previously with the BURN instruction. This instruction is valid only in the command mode and has the following form.

```
DELPRM [constant]
```

The constant value refers to a sequential program number that must be within the range of the number of available programs stored in the Archival Storage Area. The constant value can be any number in the range of 1 to 255.

Execution of the DELPRM X instruction will delete program X which was saved using the BURN instruction. The remaining programs will be compressed to save memory space, and the remaining programs will be renumbered.

Example of Use

Assume that there are five programs stored in the Archival Storage Area, and the last program's highest memory used is A000H. If program number four occupies 1000H of space, (from 9000H to A000H), and is deleted by using *DELPRM 4*, after the deletion the program that was number five will be number four, and the end of the new program four is at 9000H. If another program is saved using BURN, it will be program number five.

Error Conditions

- If Constant = < 1, or >255 ..... Constant out of range
- Constant is greater than the number of programs stored in the Archival Storage Area ..... Constant out of range
- Executed in RUN mode ..... Command mode only

GFK-0249

Commands: RAM(cr) and ROM (integer) (cr)

Action Taken:

These two commands tell the MCS BASIC-52 interpreter whether to select the current program out of RAM or the archival storage area. The current program is the one that will be displayed during a LIST command and executed when RUN is typed.

RAM

When RAM(cr) is entered, MCS BASIC-52 selects the current program from RAM memory. This is usually considered the "normal" mode of operation. It is the mode in which users interact with the command interpreter.

ROM

When ROM (integer) (cr) is entered, MCS BASIC-52 selects the current program out of the Archival Storage Area. If no integer is typed after the ROM command (i.e., ROM(cr)), MCS BASIC-52 defaults to ROM1. Since the programs are stored sequentially in the Archival Storage Area, the integer following the ROM command selects which program you want to run or list. If you attempt to select a program that does not exist (i.e., you type in ROM8 and only 6 programs are stored in the Archival Storage Area), the message 'ERROR: PROM MODE' will be displayed.

MCS BASIC-52 does not transfer the program from the Archival Storage Area to RAM when the ROM mode is selected, so you cannot edit a program in ROM. If you try to do so by typing in a line number, the message 'ERROR: PROM MODE' will be displayed. The following command to be described, XFER, allows you to transfer a program from the Archival Storage Area to RAM for editing purposes.

Since the ROM command does not transfer a program to RAM, it is possible to have different programs in ROM and RAM simultaneously. You can "flip" back and forth between the two modes at any time. Another added benefit of not transferring a program to RAM is that all of the RAM memory can be used for variable storage if the program is stored in the archival storage area. The system control values, MTOP and FREE, always refer to RAM and not the Archival Storage area.

Variation:

None.

Command: XFER(cr)

Action Taken:

The XFER (transfer) command transfers the currently selected program in the archival storage area to RAM and then selects the RAM mode. If XFER is typed while MCS BASIC-52 is in the RAM mode, the program stored in RAM is transferred back into RAM, and RAM mode is selected. The net result is that nothing happens. After the XFER command is executed, you may edit the program in the same manner any RAM program may be edited.

Variation:

None.

Command: BURN(cr)

Action Taken:

The BURN command programs the Archival Storage Area with the current selected program. The current selected program may reside in either RAM or the Archival Storage Area.

After BURN(cr) is typed, MCS BASIC-52 displays the number the program will occupy in the archival storage area.

Example:

```
>list
10      FOR I=1 TO 10
20      PRINT I
30      NEXT I

READY
>BURN
 12

READY
>ROM 12

READY
>LIST
10      FOR I=1 TO 10
20      PRINT I
30      NEXT I

READY
>
```

In this example, the program just placed in the Archival Storage Area is the twelfth program stored.

Variation:

None

GFK-0249

Statement: CLEAR

Mode: COMMAND and/or RUN

Type: CONTROL

The CLEAR statement sets all variables equal to 0 and resets all BASIC-evoked interrupts and stacks. This means that after the CLEAR statement is executed, an ONTIME statement must be executed before MCS BASIC-52 will acknowledge interrupts. ERROR trapping via the ONERR statement will also not occur until an ONERR [integer] statement is executed. The CLEAR statement does not affect the real time clock that is enabled by the CLOCK1 statement. CLEAR also does not reset the memory that has been allocated for strings, so it is *not* necessary to enter the STRING [expr], [expr] statement to reallocate memory for strings after the CLEAR statement is executed. In general, CLEAR is simply used to "erase" all variables.

Variation:

None.

Statements: CLEARI and CLEARS

Mode: COMMAND and/or RUN

Type: CONTROL

CLEARI (Clear Interrupts)

The CLEARI statement clears all the BASIC-evoked interrupts. Specifically, the ONTIME interrupt is DISABLED after the CLEARI statement is executed. The real-time clock, which is enabled by the CLOCK1 statement, is not affected by CLEARI. This statement can be used to selectively DISABLE interrupts during specific sections of your BASIC program. The ONTIME statement *must* be executed again before the specific interrupts will be enabled.

CLEARS (Clear Stacks)

The CLEARS statement resets all of MCS BASIC-52's stacks. The control and argument stacks are reset to their initialization value, 254 (OFEH) and 510 (IFEH), respectively. The internal stack (the 8052AH's stack pointer, special function register, SP) is loaded with the value that is in internal RAM location 62 (3EH). This statement can be used to "purge" the stack if an error occurs in a subroutine. In addition, this statement can be used to provide a "special" exit from a FOR-NEXT, DO-WHILE, or DO-UNTIL loop.

Example:

```
>10 PRINT "MULTIPLICATION TEST, YOU HAVE 5 SECONDS"
>20 FOR I = 2 TO 9
>30 N = INT(RND*10) : A = N*I
>40 PRINT "WHAT IS ",N,"*",I,"?": CLOCK1
>50 TIME = 0 : ONTIME 5,200 : INPUT R:IF R<>A THEN 100
>60 PRINT "THAT'S RIGHT":TIME=0:NEXT I
>70 PRINT "YOU DID IT, GOOD JOB":END
>100 PRINT "WRONG, TRY AGAIN":GOTO 50
>200 REM WASTE CONTROL STACK, TOO MUCH TIME
>210 CLEARS:PRINT "YOU TOOK TOO LONG":GOTO 10
```

NOTE

When the CLEARS and CLEARI statements are LISTED, they will appear as CLEAR S and CLEAR I, respectively.

Statements: CLOCK1 and CLOCK0

Mode: COMMAND and/or RUN

Type: CONTROL

CLOCK1

The CLOCK1 statement enables the real-time clock feature resident on the MCS BASIC-52 device. The special function operator, TIME, is incremented once every 5 milliseconds after the CLOCK1 statement has been executed. The CLOCK1 statement uses TIMER/COUNTER 0 in the 13-bit mode to generate an interrupt once every 5 milliseconds. Because of this, the special function operator, TIME, has a resolution of 5 milliseconds.

MCS BASIC-52 automatically calculates the proper reload value for TIMER/COUNTER 0 after the crystal value has been assigned (i.e. XTAL = value). If no crystal value is assigned, MCS BASIC-52 assumes a value of 11.0592 mHz. The special function operator TIME counts from 0 to 65535.995 seconds. After reaching a count of 65535.995 seconds, TIME overflows back to a count of zero.

Because the CLOCK1 statement uses the interrupts associated with TIMER/COUNTER 0 (the CLOCK1 statement sets bits 7 and 2 in the 8052AH's special function register, IE), you may not use this interrupt in an assembly language routine if the CLOCK1 statement is executed in BASIC. The interrupts associated with the CLOCK1 statement cause MCS BASIC-52 programs to run at about 99.6% of normal speed. That means that the interrupt handling for the real time clock feature only consumes about .4% of the total CPU time. This very small interrupt overhead is attributed to the very fast and effective interrupt handling of the 8052AH device.

CLOCK0

The CLOCK0 (zero) statement disables or "turns off" the real time clock feature. This statement clears BIT 2 in the 8052AH's special function register, IE. After CLOCK0 is executed, the special function operator, TIME, will no longer increment. The CLOCK0 statement also returns control of the interrupts associated with TIMER/COUNTER 0 back to the user, so this interrupt may be handled at the assembly

language level. CLOCK0 is the only MCS BASIC-52 statement that can disable the real time clock. CLEAR and CLEARI will *not* disable the real-time clock.

Variation:

None.

Statements: DATA - READ - RESTORE

Mode: RUN

Type: ASSIGNMENT

DATA

DATA specifies expressions that may be retrieved by a READ statement. If multiple expressions per line are used, they must be separated by a comma.

READ

READ retrieves the expressions that are specified in the DATA statement and assigns the value of the expression to the variable in the READ statement. The READ statement must *always* be followed by one or more variables. If more than one variable follows a READ statement, they must be separated by a comma.

RESTORE

RESTORE "resets" the internal read pointer back to the beginning of the data so that it may be read again.

Example:

```
>10 FOR I=1 TO 3
>20 READ A,B
>40 NEXT I
>50 RESTORE
>60 READ A,B
>70 PRINT A,B
>80 DATA 10,20,10/2,20/2,SIN(PI),COS(PI)
>RUN

 10  20
 5  10
 0 -1
 10  20
```

Variation:

None.

Statement: DIM

Mode: COMMAND and/or RUN

Type: ASSIGNMENT

DIM reserves storage for matrices. The storage area is first assumed to be zero. Matrices in MCS BASIC-52 may have only one dimension, and the size of the dimensioned array may not exceed 254 elements. Once a variable is dimensioned in a program, it may not be redimensioned. An attempt to redimension an array will cause an ARRAY SIZE error. If an arrayed variable is used that has not been dimensioned by the DIM statement, BASIC will assign a default value of 10 to the array size. All arrays are set equal to zero when the RUN command, NEW command, or CLEAR statement is executed. The number of bytes allocated for an array is 6 times the (array size plus 1). So, the array A(100) would require 606 bytes of storage. Memory size usually limits the size of a dimensioned array.

Variation:

More than one variable can be dimensioned by a single DIM statement (i.e., DIM A(10), B(15), A1(20)).

Example:

```
DEFAULT ERROR ON ATTEMPT TO RE-DIMENSION ARRAY

>10 A(5)=10       - BASIC assigns default of 10 TO ARRAY SIZE HERE
>20 DIM A(5)      - ARRAY CANNOT BE RE-DIMENSIONED
>RUN

ERROR: ARRAY SIZE - IN LINE 20

20    DIM A(5)
----------X
```

GFK-0249

Statements: DO - UNTIL [rel expr]

Mode: RUN

Type: CONTROL

The DO - UNTIL [rel expr] statement provides a means of "loop control" within an MCS BASIC-52 program. All statements between the DO and the UNTIL [rel expr] will be executed until the relational expression following the UNTIL statement is TRUE. DO - UNTIL statements can be nested.

Example:

```
SIMPLE DO-UNTIL

>10 A=0
>20 DO
>30 A=A+1
>40 PRINT A
>50 UNTIL A=4
>60 PRINT "DONE"
>RUN

 1
 2
 3
 4
DONE

READY
>
```

```
NESTED DO-UNTIL

>10 DO : A=A+1 : DO : B=B+1
>20 PRINT A,B,A*B
>30 UNTIL B=3
>40 B=0
>50 UNTIL A=3
>RUN

 1  1  1
 1  2  2
 1  3  3
 2  1  2
 2  2  4
 2  3  6
 3  1  3
 3  2  6
 3  3  9

READY
>
```

NOTE

It is not possible to Exit the loop by use of a GOTO statement from the middle of the loop. If it is desired to Exit the loop before it's normal termination, you must set the loop variable equal to its Exit value inside the loop.

Statements: DO - WHILE [rel expr]

Mode: RUN

Type: CONTROL

The DO - WHILE [rel expr] statement provides a means of "loop control" within an MCS BASIC-52 program. This operation of this statement is similar to the DO - UNTIL [rel expr], except that all statements between the DO and the WHILE [rel expr] will be executed as long as the relational expression following the WHILE statement is true. DO - WHILE statements can be nested.

Example:

```
SIMPLE DO-WHILE

>10 DO
>20 A=A+1
>30 PRINT A
>40 WHILE A<4
>50 PRINT "DONE"
>RUN

 1
 2
 3
 4
DONE

READY
>
```

```
NESTED DO-WHILE

>10 DO : A=A+1 : B=B+1
>20 PRINT A,B,A*B
>30 WHILEL B<>3
>40 B=0
>50 UNTIL A=3
>RUN

 1  1  1
 1  2  2
 1  3  3
 2  1  2
 2  2  4
 2  3  6
 3  1  3
 3  2  6
 3  3  9

READY
>
```

NOTE

It is not possible to Exit the loop by use of a GOTO statement from the middle of the loop. If it is desired to Exit the loop before it's normal termination, you must set the loop variable equal to its Exit value inside the loop.

Statement: END

Mode: RUN

Type: CONTROL

The END statement terminates program execution. The continue command, CONT, will not operate if the END statement is used to terminate execution (i.e., a CAN'T CONTINUE error will be printed to the console). The last statement in an MCS BASIC-52 program will automatically terminate program execution if no END statement is used.

GFK-0249

Example:

```
LAST STATEMENT TERMINATION

>10 FOR I=1 TO 4
>20 PRINT I
>30 NEXT I
>RUN

 1
 2
 3
 4

READY
>
```

```
END STATEMENT TERMINATION

>10 FOR I=1 TO 4
>20 GOSUB 100
>30 NEXT I
>40 END
>100 PRINT I
>110 RETURN
>RUN

 1
 2
 3
 4

READY
>
```

Variation:

None.

Statements: FOR - TO - {STEP} - NEXT

Type: CONTROL

FOR - TO - {STEP} - NEXT statements are used to set up and control loops.

Example:

```
>10 FOR A=B TO C STEP D
>20 PRINT A
>30 NEXT A
```

If B=0, C=10, and D=2, the PRINT statement at line 20 will be executed six times. The values of "A" that will be printed are 0, 2, 4, 6, 8, 10. "A" represents the name of the index or loop counter. The value of "B" is the starting value of the index, the value of "C" is the limit value of the index, and the value of "D" is the increment of the index. If the STEP statement and the value "D" are omitted, the increment value defaults to 1. Therefore, STEP is an optional statement. The NEXT statement causes the value of "D" to be added to the index. The index is then compared to the value of "C", the limit. If the index is less than or equal to the limit, control will be transferred back to the statement after the FOR statement. Stepping "backwards" (i.e. FOR I = 100 TO 1 STEP-1) is permitted in MCS BASIC-52. Unlike some BASICS, the index may *not* be omitted from the NEXT statement in MCS BASIC-52 (i.e., the NEXT statement *must* always be followed by the appropriate variable).

NOTE

It is not possible to Exit the loop by use of a GOTO statement from the middle of the loop. If it is desired to Exit the loop before it's normal termination, you must set the loop variable equal to its Exit value inside the loop.

Example:

```
>10 FOR I=1 TO 4
>20 PRINT I
>30 NEXT I
>RUN

 1
 2
 3
 4

READY
>
```

```
>10 FOR I=0 TO 8 STEP 2
>20 PRINT I
>30 NEXT I
>RUN

 0
 2
 4
 6
 8

READY
>
```

Statements: GOSUB [ln num] - RETURN

Mode: RUN

Type: CONTROL

GOSUB

The GOSUB [ln num] statement will cause MCS BASIC-52 to transfer control of the program directly to the line number ([ln num]) following the GOSUB statement. In addition, the GOSUB statement saves the location of the statement following GOSUB on the control stack so that a RETURN statement can be performed to return control.

RETURN

The Return statement is used to "return" control back to the statement following the most recently executed GOSUB statement. The GOSUB statement can call another subroutine with another GOSUB statement.

Example:

```
SIMPLE SUBROUTINE
>10 FOR I=1 TO 5
>20 GOSUB 100
>30 NEXT I
>100 PRINT I
>110 RETURN
>RUN

1
2
3
4
5

READY
>
```

```
NESTED SUBROUTINES
>10 FOR I=1 TO 3
>20 GOSUB 100
>30 NEXT I
>40 END
>100 PRINT I
>110 GOSUB 200
>120 RETURN
>200 PRINT I*I
>210 RETURN
>RUN

1 1
2 4
3 9

READY
>
```

GFK-0249

Statement: GOTO [ln num]

Mode: COMMAND and/or RUN

Type: CONTROL

The GOTO [ln num] statement will cause BASIC to transfer control directly to the line number ([ln num]) following the GOTO statement.

Example:

```
50 GOTO 100
```

If line 100 exists, the GOTO statement in the example above will cause execution of the program to resume at line 100. If line number 100 does not exist, the message 'ERROR: INVALID LINE NUMBER' will be printed to the console device.

Unlike the RUN command, the GOTO statement, if executed in the Command mode, does not CLEAR the variable storage space or interrupts. However, if the GOTO statement is executed in the Command mode after a line has been edited, MCS BASIC-52 will CLEAR the variable storage space and all BASIC-evoked interrupts. This is a necessity because the variable storage and the BASIC program reside in the same RAM memory, and editing a program could destroy variables.

Example:

```
>10 GOTO 100
>20 PRINT X
>RUN

ERROR:   INVALID LINE NUMBER - IN LINE 20

20    PRINT  X
--------------X
```

Statements: ON [expr] GOSUB [ln num],[ln num], . . . [ln num]
ON [expr] GOTO [ln num],[ln num], . . . [ln num]

Mode: RUN

Type: CONTROL

The value of the expression following the ON statement is the number in the line list that control will be transferred to.

Example:

```
10 ON Q GOTO 100,200,300
```

If Q were equal to 0, control would be transferred to line number 100. If Q were equal to 1, control would be transferred to line number 200. If Q were equal to 2, GOTO line 300, etc. All comments that apply to GOSUB and GOTO apply to the ON statement. If Q is less than zero, a BAD ARGUMENT error will be generated. If Q is greater than the line number list following the GOTO or GOSUB statement, a BAD SYNTAX error will be generated. The ON statement provides "conditional branching" options within the constructs of an MCS BASIC-52 program.

Statement: IF - THEN - ELSE

Mode: RUN

Type: CONTROL

The IF statement sets up a conditional test. The generalized form of the IF - THEN - ELSE statement is as follows:

```
[ln num]IF[rel expr]THEN valid STATEMENT ELSE valid STATEMENT
```

When writing a statement that has a relational expression [rel expr] with the form *IF VAR1=VAR2*, the statement will not be compiled correctly if VAR2 is I, P, or F. For example, the statement IF V=I THEN GOTO 100 will compile as if V= THEN GOTO 100. However, the statement IF V=I1 THEN GOTO 100 will compile correctly.

Example:

```
>10  IF A=100 THEN A=0 ELSE A=A+1
```

Upon execution of line 10 if A is equal to 100, then A would be assigned a value of 0. If A does not equal 100, A would be assigned a value of A+1. If it is desired to transfer control to different line numbers using the IF statement, the GOTO statement may be omitted. The following examples would yield the same results.

GFK-0249

Example:

```
>20  IF INT(A)<10 THEN GOTO 100 ELSE GOTO 200
>20  IF INT(A)<10 THEN 100 ELSE 200
```

Additionally, the THEN statement can be replaced by any valid MCS BASIC-52 statement.

Example:

```
>30  IF A<>10 THEN PRINT A ELSE 10
>30  IF A<>10 PRINT A ELSE 10
```

The ELSE statement may be omitted. If it is, control would then pass to the next statement.

Example:

```
>20  IF A=10 THEN 40
>30  PRINT A
```

In this example, IF A equals 10, then control would be passed to line number 40. If A does not equal 10, line number 30 would be executed.

Statement: INPUT

Mode: RUN

Type: INPUT/OUTPUT

The INPUT statement allows you to enter ASCII coded data from the keyboard (Send Port) during program execution (refer to Appendix E for a list of ASCII codes). One or more variables may be assigned data with a single input statement; however, these variables must be separated by a comma.

NOTE

If an application requirement is to be able to read binary data (from 0 to FFH), you may need to use the INPB instruction (see Chapter 6 for details).

For example, INPUT A,B would cause the printing of a question mark (?) on the screen as a prompt to the operator to input two numbers separated by a comma. If the operator does not enter enough data, MCS BASIC-52 would respond by outputting the message 'TRY AGAIN' to the screen.

It is sometimes desirable to disable the "Echo Back" during an INPUT statement. For example, when you press a function key, you do not want to see miscellaneous characters on the screen. Another

example, might be when a password is being entered. It is possible to disable the INPUT echo back by entering the following:

```
XBY(0F800H) = 0A5H
XBY(0F801H) = 05AH
```

To re-enable the echo back, change the above values to 0.

Example:

```
>10 INPUT A,B
>20 PRINT A,B
>RUN

?1

TRY AGAIN

?1,2
 1  2

READY
```

The INPUT statement may be written so that a descriptive prompt is printed to tell you what to type. The message to be printed is placed in quotes after the INPUT statement. If a comma appears before the first variable on the input list, the question mark prompt character will not be displayed.

Example:

```
>10 INPUT "ENTER A NUMBER"A
>20 PRINT SQR (A)
>RUN

ENTER A NUMBER
?100
 10
```

```
>10 INPUT "ENTER A NUMBER-",A
>20 PRINT SQR(A)
>RUN

ENTER A NUMBER-100
10
```

Strings can also be assigned with an INPUT statement. Strings are always terminated with the Enter key or Return key. Therefore, if more than one string input is requested with a single INPUT statement, MCS BASIC-52 will prompt with a question mark.

Example:

```
>10 STRING 110,10
>20 INPUT "NAME: "'$(1)
>30 PRINT "HI "'$(1)
>RUN

NAME: SUSAN
HI SUSAN

READY
```

```
>10 STRING 110,10
>20 INPUT "NAMES: ",$(1),$(2)
>30 PRINT "HI "'$(1)," AND "'$(2)
>RUN

NAMES: BILL
? ANN
HI BILL AND ANN

READY
```

Additionally, strings and variables can be assigned with a single INPUT statement.

Example:

```
>10 STRING 100,10
>20 INPUT "NAME(CR), AGE - ",$(1),A
>30 PRINT "HELLO "'$(1), YOU ARE ",A, "YEARS OLD"
>RUN

NAME(CR),  AGE - FRED
?15
HELLO FRED, YOU ARE 15 YEARS OLD

READY
>
```

Statement: LET

Mode: COMMAND and/or RUN

Type: ASSIGNMENT

The LET statement is used to assign a variable to the value of an expression. The generalized form of LET is:

```
LET [var] = [expr]
```

Example:

```
LET A = 10*SIN(B)/100   or
LET A = A + 1
```

Note that the = sign used in the LET statement is not an equality operator, but rather a "replacement" operator, and that the statement should be read A is replaced by A plus 1. The word LET is always optional. For example, LET A = 2 is the same as A = 2.

When LET is omitted, the LET statement is called an "IMPLIED LET." This document will use the word LET to refer to both the LET and IMPLIED LET statements.

The LET statement is also used to assign string variables. For example, LET $(1)="THIS IS A STRING" or LET $(2)=$(1).

Before strings can be assigned, the STRING [expr], [expr] statement must be executed, or else a MEMORY ALLOCATION error will occur. Special function values can also be assigned by the LET statement. For example:

```
LET XBYTE(2000H)=5AH

or

LET DBYTE(25)=XBYTE(1000)
```

Statement: ONERR [ln num]

Mode: RUN

Type: CONTROL

The ONERR[ln num] statement lets the programmer handle arithmetic errors, should they occur during program execution. Only ARITH. OVERFLOW, ARITH. UNDERFLOW, DIVIDE BY ZERO, and BAD ARGUMENT errors can be "trapped" by the ONERR statement; all other errors cannot. If an arithmetic error occurs after the ONERR statement is executed, the MCS BASIC-52 interpreter will pass control to the line following the ONERR[ln num] statement. The programmer can handle the error condition in any manner suitable to the particular application. Typically, the ONERR[ln num] statement should be viewed as an easy way to handle errors when inappropriate data is provided to an INPUT statement.

With the ONERR[ln num] statement, the programmer has the option of determining what type of error occurred. This is done by examining external memory location 257 (101H) after the error condition is trapped. The error codes are as follows:

```
ERROR CODE = 10 - DIVIDE BY ZERO
ERROR CODE = 20 - ARITH. OVERFLOW
ERROR CODE = 30 - ARITH. UNDERFLOW
ERROR CODE = 40 - BAD ARGUMENT
```

This location may be examined by using an XBY(257) statement.

Statement: ONTIME [expr],[ln num]

Mode: RUN

Type: CONTROL

Since MCS BASIC-52 processes a line in the millisecond time frame and the timer/counters on the 8052AH microcontroller operate in the microsecond time frame, there is an inherent incompatibility between the timer/counters on the 8052AH and MCS BASIC-52. To help solve this situation, the ONTIME [expr],[ln num] statement was devised.

ONTIME generates an interrupt every time the special function operator, TIME, is equal to or greater than the expression following the ONTIME statement. Only the integer portion of TIME is compared to the integer portion of the expression. The interrupt forces a GOSUB to the line number ([ln num]) following the expression ([expr]) in the ONTIME statement.

Since the ONTIME statement uses the special function operator, TIME, the CLOCK1 statement must be executed in order for ONTIME to operate. If CLOCK1 is not executed, the special function operator, TIME, will never increment and not much will happen.

In order to generate periodic interrupts, the ONTIME statement must be executed again in the interrupt routine, as shown in the following example:

Example:

```
>10  TIME=0 : CLOCK1 : ONTIME2,100 : DO
>20  WHILE TIME <10 : END
>100 PRINT "TIMER INTERRUPT AT -", TIME,"SECONDS"
>110 ONTIME TIME+2,100 : RETI
>RUN

TIMER INTERRUPT AT - 2.045 SECONDS
TIMER INTERRUPT AT - 4.045 SECONDS
TIMER INTERRUPT AT - 6.045 SECONDS
TIMER INTERRUPT AT - 8.045 SECONDS
TIMER INTERRUPT AT -10.045 SECONDS

READY
```

Note that the TIME that was printed out was 45 milliseconds greater than the time that the interrupt was supposed to be generated. That's because the terminal used in this example was running at 4800 baud and it takes about 45 milliseconds to print the message 'TIMER INTERRUPT AT -" "'.

If you do not want this delay, a variable should be assigned to the special function operator, TIME, at the beginning of the interrupt routine.

Example:

```
>10  TIME=0 : CLOCK1 : ONTIME 2,100 : DO
>20  WHILE TIME <10 : END
>100 A=TIME
>110 PRINT "TIMER INTERRUPT AT -",A,"SECONDS"
>120 ONTIME A+2,100 : RETI
>RUN

TIMER INTERRUPT AT - 2 SECONDS
TIMER INTERRUPT AT - 4 SECONDS
TIMER INTERRUPT AT - 6 SECONDS
TIMER INTERRUPT AT - 8 SECONDS
TIMER INTERRUPT AT - 10 SECONDS

READY
```

CAUTION

The ONTIME interrupt routine must be exited with a RETI statement. Failure to do this will "lock out" all future interrupts.

The ONTIME statement in MCS BASIC-52 is unique, relative to most BASICS. This powerful statement eliminates the need for you to "test" the value of the TIME operator periodically throughout the BASIC program.

GFK-0249

Statement: PRINT or P.

Mode: COMMAND and/or RUN

Type: INPUT/OUTPUT

The PRINT statement directs MCS BASIC-52 to output to the console device. The value of expressions, strings, literal values, variables or test strings may be printed out. The various forms may be combined in the print list by separating them with commas. If the list is terminated with a comma, the carriage return/line feed will be suppressed. P is a "shorthand" notation for PRINT; ? is also a "shorthand" notation for PRINT.

Example:

```
>PRINT 10*10, 3*3        >PRINT "MCS-52"        >PRINT 5,1E3
 100  9                   MCS-52                  5  1000
```

Values are printed next to one another with two intervening blanks. A PRINT statement with no arguments causes a carriage return/line feed sequence to be sent to the console device.

Special Print Formatting Statements

TAB([expr])

The TAB([expr]) function is used in the PRINT statement to cause data to be printed out in exact locations on the output device. TAB([expr]) tells MCS BASIC-52 which position to begin printing the next value in the print list. If the printhead or cursor is on or beyond the specified TAB position, MCS BASIC-52 will ignore the TAB function.

Example:

```
>PRINT  TAB(5),"X",TAB(10),"Y"
     X     Y
```

SPC([expr])

The SPC([expr]) function is used in the PRINT statement to cause MCS BASIC-52 to output the number of spaces in the SPC argument.

Example:

```
>PRINT A,SPC(5),B
```

The example above may be used to place an additional 5 spaces between the A and B, over and above the two that would normally be printed.

GFK-0249

CR

The CR function is unique to MCS BASIC-52. When CR is used in a PRINT statement, it will force a carriage return but no line feed. This can be used to create one line on a CRT device that is repeatedly updated.

Example:

```
>10 FOR I=1 TO 1000
>20 PRINT I,CR,
>30 NEXT I
```

In the example above, the output will remain only on one line. No line feed will ever be sent to the console device.

USING(special characters)

The USING function is used to tell MCS BASIC-52 what format to display the values that are printed. MCS BASIC-52 "stores" the desired format after the USING statement is executed. All outputs following a USING statement will then be in the format evoked by the last USING statement executed. The USING statement need not be executed within every PRINT statement unless the programmer wants to change the format. U. is a "shorthand" notation for USING. The options for USING as as follows:

USING(Fx): This will force MCS BASIC-52 to output all numbers using the floating point format. The value of x determines how many significant digits will be printed. If x equals 0, MCS BASIC-52 will not output any trailing zeros, so the number of digits will vary depending upon the number. MCS BASIC-52 will always output at least 3 significant digits, even if x is 1 or 2. The maximum value for x is 8.

Example:

```
>10 PRINT USING(F3),1,2,3
>20 PRINT USING(F4),1,2,3
>30 PRINT USING(F5),1,2,3
>40 FOR I=10 TO 40 STEP 10
>50 PRINT I
>60 NEXT I
>RUN

1.00 E 0  2.00 E 0  3.00 E 0
1.000 E 0  2.000 E 0  3.000 E 0
1.0000 E 0  2.0000 E 0  3.0000 E 0
1.0000 E+1
2.0000 E+1
3.0000 E+1
4.0000 E+1

READY
```

Statements: PH0., PH1.

Mode: COMMAND and/or RUN

Type: INPUT/OUTPUT

The PH0. and PH1. statements do the same thing as the PRINT statement, except that the values are printed out in a hexadecimal format. The PH0. statement suppresses two leading zeros if the number to be printed is less than 255 (0FFH). The PH1. statement always prints out four hexadecimal digits. The character "H" is always printed after the number when PH0. or PH1. is used to direct an output. The values printed are always truncated integers. If the number to be printed is not within the range of valid integers (i.e. between 0 and 65535 (0FFFFH), inclusive), MCS BASIC-52 will default to the normal mode of print. If this happens, no "H" will be printed out after the value. Since integers can be entered in either decimal or hexadecimal form, the statements PRINT, PH0., and PH1. can be used to perform decimal-to-hexadecimal and hexadecimal-to-decimal conversion. All comments that apply to the PRINT statement apply to the PH0. and PH1. statements.

Example:

```
>PH0. 2*2      >PH1. 2*2     >PRINT 99H     >PH0. 100
 04H            0004H         153            64H

>PH0. 1000     >PH1. 1000    >P. 3E8H       >PH0. PI
 3E8H           03E8H         1000           03H
```

Statement: PUSH [expr]

Mode: COMMAND and/or RUN

Type: ASSIGNMENT

Arithmetic expressions following the PUSH statement are evaluated and then sequentially placed on MCS BASIC-52's argument stack. The PUSH statement, in conjunction with the POP statement, provides a simple means of passing parameters to assembly language routines. In addition, the PUSH and POP statements can be used to pass parameters to BASIC subroutines and to "SWAP" variables. The last value "pushed" onto the argument stack will be the first value "popped" off the argument stack.

Variation:

More than one expression can be pushed onto the argument stack with a single PUSH statement. The expressions are simply followed by a comma, (i.e., PUSH[expr],[expr], ... [expr]). The last value pushed onto the argument stack will be the last expression ([expr]) encountered in the PUSH statement.

GFK-0249

Example:

```
SWAPPING                SUBROUTINE
VARIABLES               PASSING
>10 A=10                >10 PUSH 1,3,2
>20 B=20                >20 GOSUB 100
>30 PRINT A,B           >30 POP R1,R2
>40 PUSE A,B            >40 PRINT R1,R2
>50 POP A,B             >50 END
>60 PRINT A,B           >100 REM QUADRATIC A=2,B=3,C=1 IN EXAMPLE
                        >110 POP A,B,C
                        >120 PUSH (-B+SQR(B*B-4*A*C))/(2*A)
10 20                   >130 PUSH (-B-SQR(B*B-4*A*C))/(2*A)
20 10                   >140 RETURN
                        >RUN
READY
>                       -1-5
                        READY
                        >
```

Statement: POP [var]

Mode: COMMAND and/or RUN

Type: ASSIGNMENT

The top of the argument stack is assigned to the variable following the POP statement, and the argument stack is "popped" (i.e., incremented by 6). Values can be placed on the stack by using either the PUSH statement or assembly language CALLS. However, if a POP statement is executed and no number is on the argument stack, an A-STACK error will occur.

Variation:

More than one variable can be popped off the argument stack with a single POP statement. The variables are simply followed by a comma (i.e., POP[var],[var],.....[var]).

Example:

For an example, refer to the one shown above for the PUSH statement.

Comment:

The PUSH and POP statements are unique to MCS BASIC-52. These powerful statements can be used to "get around" the global variable problems so often encountered in BASIC programs. This problem arises in BASIC because the "main" program and all subroutines used by the main program are required to use the same variable names (i.e., global variables). It is not always convenient to use the same variables in a subroutine as in the main program, and you often see programs reassign a number of variables (i.e., A=Q) before a GOSUB statement is executed. If you reserve some variable names *just* for subroutines (i.e., S1, S2) and pass variables on the stack, as shown in the previous example, you will avoid any global variable problems in MCS BASIC-52.

Statement: REM

Mode: COMMAND AND/OR RUN

Type: CONTROL - (performs no operation)

REM is short for REMark. It does nothing, but allows you to add comments to a program. Comments are usually needed to make a program easier to understand. A REM statement may not be terminated by a colon (:) ; however, it may be placed after a colon. This can be used to allow the programmer to place a comment on each line.

Example:

```
>10 REM INPUT ONE VARIABLE
>20 INPUT A
>30 REM INPUT ANOTHER VARIABLE
>40 INPUT B
>50 REM MULTIPLY THE TWO
>60 Z=A*B
>70 REM PRINT THE ANSWER
>80 PRINT Z


>10 INPUT A: REM INPUT ONE VARIABLE
>20 INPUT B : REM INPUT ANOTHER VARIABLE
>30 Z=A*B : REM MULTIPLY THE TWO
>40 PRINT Z : REM PRINT THE ANSWER
```

The following will *not* work because the entire line would be interpreted as a REMark, so the PRINT statement would not be executed.

```
>10 REM PRINT THE NUMBER : PRINT A
```

NOTE

The REM statement is executable in the Command mode of MCS BASIC-52 so that you can use an UPLOAD/DOWNLOAD routine. This allows you to insert REM statements, without line numbers in the text, and not download them to the MCS BASIC-52 device; and also helps to conserve memory.

Statement: RETI

Mode: RUN

Type: CONTROL

The RETI statement is used to exit from interrupts that are handled by an MCS BASIC-52 program (specifically, ONTIME interrupts). The RETI statement does the same thing as the RETURN statement, except that it also clears software interrupt flags so interrupts can again be acknowledged. If you fail to execute the RETI statement in the interrupt procedure, all future interrupts will be ignored.

GFK-0249

Statement: STOP

Mode: RUN

Type: CONTROL

The STOP statement allows the programmer to break program execution at specific points in a program for easy program "debugging". After a program is STOPped, variables can be displayed and/or modified. Program execution may be resumed with a CONTinue command.

Example:

```
>10 FOR I=1 to 100
>20 PRINT I
>30 STOP
>40 NEXT I
>RUN

 1
STOP - IN LINE 40

READY
>CONT

 2
```

NOTE

The line number printed out after the STOP statement is executed is the line number following the STOP statement, not the line number that contains the STOP statement.

Statement: STRING [expr],[expr]

Mode: COMMAND and/or RUN

Type: CONTROL

The STRING [expr],[expr] statement allocates memory for strings. Initially, no memory is allocated for strings. If you try to define a string with a statement, such as LET $(1)="HELLO", before memory has been allocated for strings, a MEMORY ALLOCATION error will be generated. The first expression in the STRING [expr], [expr] statement is the total number of bytes you wish to allocate for string storage. The second expression denotes the maximum number of bytes in each string. These two numbers determine the total number of defined string variables.

You might think that the total number of defined strings would be equal to the first expression in the STRING [expr],[expr] statement divided by the second expression. However, MCS BASIC-52 requires one additional byte for each string, plus one additional byte overall. This means that the statement STRING 100.10 would allocate enough memory for 9 string variables, ranging from $(0) to $(8), and all of the 100 allocated bytes would be used. Note that $(0) is a valid string in MCS BASIC-52.

After memory is allocated for string storage, neither commands, such as NEW, nor statements, such as CLEAR, will "deallocate" this memory. The only way memory can be deallocated is to execute a STRING 0,0 statement. STRING 0,0 will allocate no memory to string variables.

NOTE

Every time the STRING [expr],[expr] statement is executed, MCS BASIC-52 executes the equivalent of a CLEAR statement. This is necessary because string variables and numeric variables occupy the same external memory space. After the STRING statement is executed, all variables are "wiped-out". Because of this, string memory allocation should be performed early in a program. String memory should never be "reallocated" unless the programmer is willing to destroy all defined variables.

Statement: IDLE

Mode: RUN

Type: CONTROL

The IDLE statement forces the MCS BASIC-52 device into a "wait until interrupt" mode. Execution of statements is halted until an ONTIME [expr], [ln num] interrupt is received. You must make sure that this interrupt has been enabled before executing the IDLE instruction, or else the MCS BASIC-52 device will enter a "wait forever" mode and the system will crash.

When an ONTIME [expr], [ln num] interrupt is received while in the IDLE mode, the MCS BASIC-52 device will execute the interrupt routine and then the statement following the IDLE instruction. Thus, the execution of the IDLE instruction is terminated when an interrupt is received.

An attempt to execute the IDLE statement in the Direct mode will yield a BAD SYNTAX error.

Statement: RROM (integer)

Mode: COMMAND and/or RUN

Type: CONTROL

RROM stands for RUN ROM. What it does is select a program in the Archival Storage Area and then execute the program. The integer after the RROM statement selects which program in the Archival Storage Area is to be executed. In the Command mode, RROM 2 would be equivalent to typing ROM 2, then RUN. But, notice that RROM [integer] is a statement. This means that a program that is already executing can actually force the execution of a completely different program than is in the Archival Storage Area. This allows you to "change programs" on the fly.

GFK-0249

If you execute a RROM [integer] statement and an invalid integer is entered (i.e., 6 programs are contained in the Archival Storage Area and you enter RROM 8), no error will be generated and MCS BASIC-52 will execute the statement following the RROM [integer] statement.

NOTE

Every time the RROM [integer] statement is executed, all variables and strings are set equal to zero. Variables and strings cannot be passed from one program to another by using the RROM [integer] statement. Additionally, all MCS BASIC-52 evoked interrupts are cleared. Variables can be passed to new programs, however, by storing them in the PSA and using the ST@ and LD@ statements.

Statements: ST@ [expr] and LD@ [expr]

Mode: COMMAND and/or RUN

Type: INPUT/OUTPUT

ST@

The ST@ [expr] statement allows you to specify where MCS BASIC-52 floating point numbers are to be stored. The expression ([expr]) following the ST@ statement specifies the address of where the number is to be stored and the number is assumed to be on the argument stack. The ST@ [expr] statement is designed to be used in conjunction with the LD@ [expr] statement. The purpose of these two statements is to allow you to save floating point numbers in the Permanent Storage Area.

LD@

The LD@ [expr] statement allows you to retrieve floating point numbers that were saved with the ST@ [expr] statement. The expression ([expr]) following the LD@ statement specifies where the number is stored. After executing the LD@ [expr] statement, the number is placed on the argument stack.

Example:

Saving and retrieving a 10-element array at location array at location 0F000H:

```
10  REM *** ARRAY SAVE ***
20  FOR I=0 TO 9
30  PUSH A(I): REM PUT ARRAY VALUE ON STACK
40  ST@ 0F005H+6*I: REM STORE IT, SIX BYTES PER NUMBER
50  NEXT I
60  REM *** GET ARRAY ***
70  FOR I=0 TO 9
80  LD@ 0F005H+6*I
90  POP B(I)
100 NEXT I
```

Remember that each floating point number requires 6 bytes of storage. Also, note that expressions in the ST@ [expr] and LD@ [expr] statements point to the most significant byte of the stored number. Therefore, ST@ (0F005H) would save the number in locations 0F005H, 0F004H, 0F003H, 0F002H, 0F001H, and 0F000H.

Chapter 9
Operators and Expressions

Dual Operand Operators

+ ADDITION OPERATOR

Example:

```
print 3+2
5
```

/ DIVISION OPERATOR

Example:

```
PRINT 100/5
20
```

**EXPONENTIATION OPERATOR

Raises the first expression to the power of the second expression. The power any number can be raised to is limited to 255. To eliminate confusion, the notation ** was chosen instead of the Up arrow symbol, because the Up arrow is displayed differently on various terminals.

Example:

```
PRINT 2**3
8
```

* MULTIPLICATION OPERATOR

Example:

```
PRINT 3*3
9
```

- SUBTRACTION OPERATOR

Example:

```
PRINT 9-6
3
```

.AND. LOGICAL AND OPERATOR

Example:

```
PRINT 3.AND.2
2
```

.OR. LOGICAL OR OPERATOR

Example:

```
PRINT 1.OR.4
5
```

.XOR. LOGICAL EXCLUSIVE OR OPERATOR

Example:

```
PRINT 7.XOR.6
1
```

Comments on Logical Operators .AND., .OR., and .XOR.

These operators perform a bit-wise logical function on valid integers. That means both arguments for these operators must be between 0 and 65535 (0FFFH), inclusive. If they are not, MCS BASIC-52 will generate a BAD ARGUMENT error. All non-integer values are truncated, i.e., NOT rounded.

MCS BASIC-52 eliminates *all* spaces when it processes a user line, and inserts spaces before and after statements when it lists a user program. MCS BASIC-52 does not insert spaces before and after operators. So, if you type in a line such as 10 A = 10 * 10, the line will be listed as 10 A = 10*10. All spaces entered before and after the operator will be eliminated.

The .OP. notation was chosen for the logical operators because a line entered as 10 B = A AND B would be listed as 10 B=AANDB. This looked confusing, so dots were added to the logical instructions. The previous example would be listed as 10 B=A.AND.B, which is easier to read.

GFK-0249

Single Operand Operators - General Purpose

ABS([expr])

Returns the absolute value of the expression.

Example:

```
PRINT ABS(5)          PRINT ABS (-5)
5                     5
```

NOT([expr])

Returns a 16 bit one's complement of the expression. The expression must be a valid integer between 0 and 65535 (0FFFFH), inclusive. Non-integers will be truncated, not rounded.

Example:

```
PRINT NOT(65000)          PRINT NOT(0)
535                       65535
```

INT([expr])

Returns the integer portion of the expression.

Example:

```
PRINT INT(3. 7)           PRINT INT(100, 876)
3                         100
```

SGN([expr])

Returns a value of +1 if the argument is greater than zero, zero if the argument is equal to zero, and -1 if the argument is less than zero.

Example:

```
PRINT SGN(52)          PRINT SGN(0)          PRINT SGN(-8)
1                      0                     -1
```

SQR([expr])

Returns the square root of the argument. The argument may not be less than zero. The result returned will be accurate within ± a value of 5 on the least significant digit.

Example:

```
PRINT SQR(9)        PRINT SQR(45)        PRINT SQR(100)
3                   6.7082035            10
```

RND

Returns a pseudo-random number in the range between 0 and 1, inclusive. The RND operator uses a 16-bit binary seed and generates 65536 pseudo-random numbers before repeating the sequence. Unlike most BASICS, the RND operator in MCS BASIC-52 does not require an argument or a dummy argument. In fact, if an argument is placed after RND operator, a BAD SYNTAX error will occur.

Example:

```
PRINT RND
 30278477
```

PI

PI is not really an operator; it is a stored constant. In MCS BASIC-52, PI is stored as 3.1415926. Math experts will notice that PI is actually closer to 3.141592653, so proper rounding for PI should yield the number 3.1415927. The reason MCS BASIC-52 uses a 6 instead of a 7 for the last digit is that errors in the SIN, COS and TAN operators were found to be greater when the 7 was used instead of 6. This is because the number PI/2 is needed for these calculations and it is desirable, for the sake of accuracy, to have the equation PI/2 + PI/2 = PI hold true. This cannot be done if the last digit in PI is an odd number, so the last digit of PI was rounded to 6 instead of 7 to make these calculations more accurate.

Single Operand Operators - Log Functions

LOG([expr])

Returns the natural logarithm of the argument. The argument must be greater than 0. This calculation is carried out to 7 significant digits.

Example:

```
PRINT LOG (12)        PRINT LOG (EXP(1))
2.484906              1
```

GFK-0249

EXP([expr])

This function raises the number "e" (2.7182818) to the power of the argument.

Example:

```
PRINT EXP(1)          PRINT EXP(LOG(2))
2.7182818             2
```

Single Operand Operators - Trig Functions

SIN([expr])

Returns the SIN of the argument. The argument is expresses in radians. Calculations are carried out to 7 significant digits. The argument must be between ±1200000.

Example:

```
PRINT SIN(PI/4)          PRINT SIN(0)
.7071067                 0
```

COS([expr])

Returns the COS of the argument. The argument is expressed in radians. Calculations are carried out to 7 significant digits. The argument must be between ±200000.

Example:

```
PRINT COS(PI/4)          PRINT COS(0)
.7071067                 1
```

TAN([expr])

Returns the TAN of the argument. The argument is expressed in radians. The argument must be between ±1200000.

Example:

```
PRINT TAN(PI/4)          PRINT TAN(0)
1                        0
```

ATN([expr])

Returns the ARCTANGENT of the argument. The result is in radians. Calculations are carried out to 7 significant digits. The ATN operator returns a result between -PI/2 (3.1415926/2) and PI/2.

Example:

```
PRINT ATN(PI)          PRINT ATN(1)
1.2626272              .78539804
```

Precedence of Operators

The hierarchy of mathematics dictates that some operations are carried out before others. If you understand this hierarchy, it is possible to write complex expressions using only a minimum amount of parentheses. It is easy to illustrate what precedence is all about by examining the following equation:

```
4+3*2= ?
```

Should you add (4+3), and then multiply 7 by 2, or should you multiply (3*2), and then add 4? The hierarchy of mathematics says that multiplication has precedence over addition, so you would multiply (3*2) first, and then add 4. So,

```
4+3*2=10
```

The rules for hierarchy of math are simple. When an expression is scanned from left to right, an operation is not performed until an operator of lower or equal precedence is encountered. In the previous example, addition could not be performed because multiplication has higher precedence. The precedence of operators from highest to lowest in MCS BASIC-52 is as follows:

1. Operators that use parentheses ( )
2. Exponentiation (**)
3. Negation (-)
4. Multiplication (*) and Division (/)
5. Addition (+) and Subtraction (-)
6. Relational Expressions (=, <>, >, >=, <, <=)
7. Logical AND (.AND.)
8. Logical OR (.OR.)
9. Logical XOR (.XOR.)

NOTE

When in doubt over which operator takes precedence, use parentheses.

How Relational Expressions Work

Relational expressions involve the operators =, >, >=, <, <=, and <>. These operators are typically used to "test" a condition. In MCS BASIC-52, relational operators return a result of 65535 (0FFFFH) if the relational expression is true, and a result of 0 if the relation expression is false. The result is returned to

the argument stack. Because of this, it is possible to actually display the result of a relational expression.

Example:

```
PRINT 1=0      PRINT 1>0      PRINT A<>A      PRINT A=A
0              65535          0               65535
```

Having a relational expression return a result offers a unique benefit in that relational expressions can actually be "chained" together using the logical operators .AND., .OR., and .XOR.. This makes it possible to test a rather complex condition with one statement.

Example:

```
>10 IF A<B.AND.A>C.OR.A>D THEN.....
```

Additionally, the NOT([expr]) operator can be used.

Example:

```
>10 IF NOT(A>B).AND.A<C THEN.....
```

By "chaining" together relational expressions with logical operators, it is possible to test very particular conditions with one statement. When using logical operators to link together relational expressions, it is important that the programmer pays careful attention to the precedence of operators. The logical operators were assigned lower precedence, relative to relational expressions, just to make the linking of relational expressions possible without using parentheses.

Chapter 10 String Operators

ASC( )

The ASC( ) operator returns the integer value of the ASCII character placed in parentheses.

Example:

```
>PRINT ASC(A)
 65
```

65 is the decimal representation for the ASCII character "A." In addition, individual characters in a predefined ASCII string can be evaluated with the ASC( ) operator.

Example:

```
>10 $(1)="THIS IS A STRING"
>20 PRINT $ (1)
>30 PRINT ASC($(1),1)
>RUN

 THIS IS A STRING
 84
```

When the ASC( ) operator is used as shown above, the $(expr) denotes what string is being accessed. The expression after the comma identifies an individual character in the string. In the above example, the first character in the string was picked out and 84 is the decimal representation for the ASCII character "T."

Example:

```
>10 $(1)="ABCDEFGHIJKL"
>20 FOR X=1 TO 12
>30 PRINT ASC($(1),X),
>40 NEXT X
>RUN

 65  66  67  68  69  70  71  72  73  74  75  76
```

The numbers printed in the previous example are the values that represent the ASCII characters A, B, C, ... L.

Additionally, the ASC( ) operator can be used to change individual characters in a defined string.

Example:

```
>10 $(1)="ABCDEFGHIJKL"
>20 PRINT $(1)
>30 ASC($(1),1)=75
>40 PRINT $(1)
>50 ASC($(1),2)=ASC($(1),3)
>60 PRINT $(1)
>RUN

ABCDEFGHIJKL
KBCDEFGHIJKL
KCCDEFGHIJKL
```

In general, the ASC( ) operator lets the programmer manipulate individual characters in a string. A simple program can determine if two strings are identical.

NOTE

The GE Fanuc - NA custom command MATCH can also be used to determine if strings are identical.

Example:

```
>10 $(1)="SECRET" : REM SECRET IS THE PASSWORD
>20 INPUT "WHAT'S THE PASSWORD - ",$(2)
>30 FOR I=1 TO 6
>40 IF ASC8$(1),I)=ASC($(2),I) THEN NEXT I ELSE 70
>50 PRINT "YOU GUESSED IT!"
>60 END
>70 PRINT "WRONG, TRY AGAIN" : GOTO 20
>RUN

WHAT'S THE PASSWORD - SECURE
WRONG, TRY AGAIN
WHAT'S THE PASSWORD - SECRET
YOU GUESSED IT
```

CHR( )

The CHR( ) operator is the converse of the ASC( ) operator. It converts a numeric expression to an ASCII character.

Example:

```
>PRINT CHR(65)
A
```

Like the ASC( ) operator, the CHR( ) operator can also identify individual characters in a defined ASCII string.

GFK-0249

Example:

```
>10 $(1)="MCS BASIC-52"
>20 FOR I"1 TO 12 : PRINT CHR($(1),I), : NEXT I
>30 PRINT : FOR I=12 TO 1 STEP -1
>40 PRINT CHR($(1),I), : NEXT I
>RUN

MCS BASIC-52
25-CISAB SCM
```

In the above example, the expressions contained within the parentheses, following the CHR operator have the same meaning as the expressions in the ASC( ) operator.

Unlike the ASC( ) operator, the CHR( ) operator *cannot* be assigned a value. A statement such as CHR($(1),1) = H, is INVALID and will generate a BAD SYNTAX error. Use the ASC( ) operator to change a value in a string. The CHR( ) operator can only be used within a PRINT statement.

Chapter 11
Special Function Operators

CBY([expr])

The CBY([expr]) operator is used to retrieve data from the PROGRAM or CODE MEMORY address space of the 8052AH microcontroller. Since CODE memory cannot be written into on the 8052AH, the CBY([expr]) operator cannot be assigned a value. It can only be read.

Example:

A = CBY(1000) causes the value in code memory space 1000 to be assigned to variable A. The argument for the CBY([expr]) operator *must* be a valid integer (i.e., between 0 and 65535 (OFFFFH)). If it is not, a BAD ARGUMENT error will occur.

DBY([expr])

The DBY([expr]) operator is used to retrieve or assign a value to the 8052AH's internal data memory. Both the value and argument in the DBY operator must be between 0 and 255, inclusive. This is because there are only 256 internal memory locations in the 8052AH microcontroller, and one byte can only represent a quantity between 0 and 255, inclusive.

Example:

```
A=DBY(B) and DBY(250) = CBY(1000)
```

The first example would assign variable A the value that is in internal memory location B. B would have to be between 0 and 255. The second example would load internal memory location 250 with the same value that is in program memory location 1000.

XBY([expr])

The XBY([expr]) operator is used to retrieve or assign a value to the 8052AH's external data memory. The argument in the XBY([expr]) operator must be a valid integer between 0 and 65535 (0FFFFH). The value assigned to the XBY ([expr]) operator must be between 0 and 255. If it is not, a BAD ARGUMENT error will occur.

Example:

```
XBY(4000H)=DBY(100) and A=XBY(0F000H)
```

GET

The GET operator only produces a meaningful result when used in Run mode. It will always return a result of zero in the Command mode. GET reads the console input device by taking a "snapshot" of it.

If a character is available from the console device, the value of the character will be assigned to GET. After GET is read in the program, GET will be assigned the value of zero until another character is sent from the console device. The following example will print the decimal representation of any character sent from the console.

Example:

```
>10 A=GET
>20 IF A<>0 THEN PRINT A
>30 GOTO 10
>RUN

65      (TYPE "A" ON CONSOLE)
49      (TYPE "1" ON CONSOLE)
24      (TYPE "CONTROL-X" ON CONSOLE)
50      (TYPE "2" ON CONSOLE)
```

The GET operator can be read only once before it is assigned a value of zero in order to guarantee that the first character entered will always be read, independent of where the GET operator is placed in the program.

TIME

The TIME operator is used to retrieve and/or assign a value to the real-time clock resident in MCS BASIC-52. After reset, TIME is equal to 0. The CLOCK1 statement enables the real-time clock. When the real-time clock is enabled, the special function operator, TIME, will increment once every 5 milliseconds. The unit of TIME is seconds, and the appropriate XTAL value must be assigned to ensure that the TIME operator is accurate.

When TIME is assigned a value with a LET statement (i.e. TIME=100), only the integer portion of TIME will be changed.

To assign the fractional portion of time, use

```
DBY(71)=0: REM SET FRACTIONAL TIME TO 0
```

NOTE

TIME will not update in Command mode. It will only update in the RUN mode.

GFK-0249

Example:

```
>CLOCK1              (enable REAL TIME CLOCK)
>CLOCK0              (disable REAL TIME CLOCK)
>PRINT TIME          (display TIME)
 3.3315
>TIME=0              (set TIME=0)
>PRINT TIME          (display TIME)
 .315                (only the integer is changed)

The "fraction" portion of TIME can be changed by manipulating
the contents of internal memory location 71 (47H).  This is
accomplished by a DBY(71) statement.  Note that each count in
internal memory location 71 (47H) represents 5 milliseconds of TIME.

Continuing with the example ...

>DBY(71)=0 (fraction of TIME=0)

>PRINT TIME
 0

>DBY(71)=3 (fraction of TIME=3, 15 ms)

>PRINT TIME
 1.5 E-2
```

XTAL

The XTAL operator tells MCS BASIC-52 at what frequency the system is operating. The XTAL operator is used by MCS BASIC-52 to calculate the real-time clock reload value, the PROM programming timing, and the software serial port baud rate generation. The XTAL value is expressed in Hz.

Example:

```
XTAL = 9000000
```

would set the XTAL value to 9 mHz.

System Control Values

MTOP

After reset, MCS BASIC-52 sizes the external memory and assigns the last valid memory address to the system control value, MTOP. MCS BASIC-52 will not use any external RAM memory beyond the value assigned to MTOP. If you wish to allocate some external memory for an assembly language routine the LET statement can be used (e.g. MTOP = USER ADDRESS). If you assign a value to MTOP that is greater than the last valid memory address, a MEMORY ALLOCATION error will be generated.

Example:

```
>PRINT MTOP
 2047

>MTOP=2000

>PRINT MTOP
 2000
```

LEN

The system control value, LEN, tells you how many bytes of memory the current selected program occupies. Obviously, LEN cannot be assigned a value, it can only be read. A NULL program (i.e., no program) will return a LEN of 1. The 1 represents the end of program file character.

FREE

The system control value, FREE, tells how many bytes of RAM memory are available to the user. When the current selected is in RAM memory, the following relationship will always hold true.

```
FREE = MTOP - LEN - 511
```

NOTE

Unlike some BASICS, MCS BASIC-52 does not require any "dummy" arguments for the system control values.

Chapter 12
Error Messages

GFK-0249

MCS BASIC-52 has a relatively sophisticated error processor. When BASIC is in Run mode, the generalized from of the error message is as follows:

```
ERROR: XXX - IN LINE YYY
YYY BASIC STATEMENT
------------X
```

where XXX is the error type and YYY is the line number of the program in which the error occurred. A specific example is:

```
ERROR: BAD SYNTAX - IN LINE 10
10 PRINT 34*21*
--------------X
```

The X signifies approximately where the error occurred in the line number. The specific location of the X may be off by one or two characters or expressions, depending on the type of error and where the error occurred in the program. If an error occurs in Command mode, only the error type will be printed out, *not* the line number. (There are no line numbers in Command mode.) The error types are described below:

BAD SYNTAX

A BAD SYNTAX error means that either an invalid MCS BASIC-52 command, statement, or operator was entered and BASIC cannot process the entry. You should check to make sure that everything was typed in correctly. A BAD SYNTAX error is also generated if the programmer attempts to use a reserved keyword as part of a variable.

BAD ARGUMENT

When the argument of an operator is not within the limits of the operator, a BAD ARGUMENT error will be generated. For example, DBY(257) would generate a BAD ARGUMENT error because the argument for the DBY operator is limited to the range 0 to 255. Similarly, XBY(5000H)=-1 would generate a BAD ARGUMENT error because the value of the XBY operator is limited to the range 0 to 255.

ARITH. UNDERFLOW

If the result of an arithmetic operation exceeds the lower limit of an MCS BASIC-52 floating point number, an ARITH. UNDERFLOW error will occur. The smallest floating point number in MCS BASIC-52 is ±1E-127. For example, 1E-80/1E+80 would cause an ARITH. UNDERFLOW error.

ARITH. OVERFLOW

If the result of an arithmetic operation exceeds the upper limit of an MCS BASIC-52 floating point number, an ARITH. OVERFLOW error will occur. The largest floating point number in MCS BASIC-52 is 1.99999999E+127. For example, 1E+70*1E+70 would cause an ARITH. OVERFLOW error.

DIVIDE BY ZERO

If a division by zero is attempted (i.e., 12/0), it will produce a DIVIDE BY ZERO error.

NO DATA

If a READ statement is executed and no DATA statement exists, or all DATA has been read and a RESTORE instruction was not executed, the message 'ERROR: NO DATA - IN LINE XXX' will be printed to the console device.

CAN'T CONTINUE

Program execution can be either by typing Ctrl C on the keyboard or by executing a STOP statement. Normally, program execution can be resumed by typing in the CONT command. However, if you edit the program after halting execution and then enter the CONT command, a CAN'T CONTINUE error will be generated. A Ctrl C must be typed during program execution or a STOP statement must be executed before the CONT command will work.

PROGRAMMING

If an error occurs while the MCS BASIC-52 device is programming, a PROGRAMMING error will be generated. An error encountered during programming destroys the archival storage area structure, so you cannot save any more programs on that particular archival storage area once a PROGRAMMING error occurs.

A-STACK

An A-STACK (ARGUMENT STACK) error occurs when the argument stack pointer is forced "out of bounds". 158 bytes of external memory are allocated for the control stack, FOR-NEXT loops require 17 bytes of control stack DO-UNTIL, DO-WHILE, and GOSUB require 3 bytes of control stack. This means that 9 nested FOR-NEXT loops is the maximum that MCS BASIC-52 can handle because 9 times 17 equals 153.

If you attempt to use more control stack than is available in MCS BASIC-52, a C-STACK error will be generated. In addition, C-STACK errors will occur if a RETURN is executed before a GOSUB, a WHILE or UNTIL before a DO, or a NEXT before a FOR.

I-STACK

An I-STACK (INTERNAL STACK) error occurs when MCS BASIC-52 does not have enough stack space to evaluate an expression. Normally, I-STACK errors will not occur unless insufficient memory has been allocated to the 8052AH's stack pointer.

ARRAY SIZE

If an array is dimensioned by a DIM statement and you attempt to access a variable outside of the dimensioned bounds, an ARRAY SIZE error will be generated.

Example:

```
>DIM A(10)
>PRINT A(11)

ERROR: ARRAY SIZE
READY
```

MEMORY ALLOCATION

MEMORY ALLOCATION errors are generated when you attempt to access strings that are "outside" the defined string limits. Additionally, if the system control value, MTOP, is assigned a value that does not contain any RAM memory, a MEMORY ALLOCATION error will occur.

Chapter 13 Application Examples

Example 1 - Transferring Data Between the ASCII/BASIC Module and the Series One/One Plus CPU

Data sent from the ASCII/BASIC Module to the Series One or One Plus CPU is scanned and set the same as any other I/O points by the CPU. However, the ASCII/BASIC Module has a specific command for reading and writing data to the CPU. This command is the TRANSFER command.

Using the Transfer Command

The TRANSFER command reads or writes data at the beginning of a CPU scan. It returns a status flag that confirms that data has been transferred to or from the CPU. Any other method of communication between the module and the CPU cannot guarantee that the CPU has had enough time to write or read correct data on the I/O lines.

The TRANSFER command has four parameters which are listed below.

Table 13-1. TRANSFER Command Parameters

Parameter	Description
R or W	Read Data - CPU to ASCII/BASIC Module Write Data - ASCII/BASIC Module to CPU
FORMAT	Specifies format of data to be transferred. Floating Point data in the ASCII/BASIC Module can be converted to BCD when transferring to the CPU. BCD data in the CPU can be converted to floating point data when transferring to the ASCII/BASIC Module. The data value to be converted must not exceed 99 in either the CPU or the ASCII/BASIC Module. If FORMAT=2, conversion takes place. If FORMAT=4, no conversion is done.
ABMLOC	The variable name of the scalar or array number where data is to be read from or written to. This variable must be defined before being used in the TRANSFER instruction.
FLAG	A scalar or array variable name which indicates the status of the TRANSFER command. There are four possible values that can be returned from the TRANSFER command. Flag = -1, transfer not complete Flag = 0, transfer completed successfully Flag = 2, ABMLOC not yet defined Flag = 3, range error in data to be transferred Flag = none of the above - Error condition

Example of TRANSFER Command

An example of a TRANSFER command is shown below, with the supporting code needed for the parameters. The example sends a value of 55 from the ASCII/BASIC Module to the CPU with the

ASCII/BASIC Module converting the value to BCD for use by the CPU. The program waits until it is completed before continuing, or stops if an error is detected.

```
10  FLAG=0
20  CODE=55
30  TRANSFER W,4,CODE,FLAG
40  IF FLAG=0 THEN GOTO 60
50  IF FLAG>0 THEN PRINT FLAG : STOP ELSE GOTO 40
60  REM program continues
```

Transferring Register Data

A convenient use for the ASCII/BASIC Module is to get register data from the CPU. Most register data is 16 bits, which requires at least two TRANSFER commands to be issued. Since there is no specific program to get register information, a program must be developed which ensures that the ASCII/BASIC Module gets the right information. BASIC code must be written in connection with relay ladder logic that will synchronize the two processors. In the BASIC program shown below, one of two codes is transferred from the ASCII/BASIC Module to indicate to the CPU which eight bits of the sixteen bit register to output for the module to read.

The data to be sent is determined by the relay ladder logic. The first step is to define all of the variables. The first code sent by the ASCII/BASIC Module is 0F4 (hexadecimal). The number must not be converted by the ASCII/BASIC Module since it is greater than 99. This code signals the CPU to put the least significant byte of the sixteen bit register out for the ASCII/BASIC Module to read after sending an acknowledge flag back verifying that it saw the request. After the data is received, a second flag is put up asking for the most significant byte in the same manner. The result is then printed to a terminal.

Code for Register Data Transfer

```
 10  LSB=0 : FLAG=0
 20  CODE=0F4H
 30   TRANSFER W,2,CODE,FLAG
 40   IF FLAG=0 THEN GOTO 60
 50   IF FLAG.0 THEN PRINT FLAG ; STOP ELSE GOTO 40
 60   TRANSFER R,4,LSB,FLAG
 70   IF FLAG =0 THEN GOTO 90
 80   IF FLAG>0 THEN PRINT FLAG ; STOP ELSE GOTO 70
 90  CODE=0F5H
100   TRANSFER W,2,CODE,FLAG
110   IF FLAG=0 THEN GOTO 130
120   IF FLAG>0 THEN PRINT FLAG : STOP ELSE GOTO 110
130   TRANSFER R,4,MSB,FLAG
140   IF FLAG=0 THEN GOTO 160
150   IF FLAG>0 THEN PRINT FLAG : STOP ELSE GOTO 140
160  COUNTER=((MSB*100)+LSB)/10
170   PRINT COUNTER
180   END
```

Relay Ladder Logic for Register Data Transfer

Relay ladder logic for this BASIC program is shown below. If you were printing the transferred data to an OIT screen at 19.2K Baud, the update time would be approximately 0.1 seconds (100 ms).

```
02/14/89     12:06:26     GE FANUC LOGICMASTER 1F documentation    Page   1
                              TRANSFER TO/FROM ABM
                              SERIES ONE PLUS CPU


 << RUNG    0 >>

 ********************************************************************************
 *                                                                              *
 *   This relay ladder diagram will transfer the value of Timer 600 to          *
 *   the ASCII/BASIC Module located in slot 1 next to the Series One            *
 *   Plus CPU.  The ASCII/BASIC Module requests the data with a series          *
 *   of flags to the CPU.  The least significant byte is transferred            *
 *   first, then the most significant byte.                                     *
 *                                                                              *
 *   For comment purposes, the following assignments have been made.            *
 *                                                                              *
 *   Registers 500 and 501 = WORK_REG                                           *
 *   Inputs 0-7 (R0) = ABM_TO_CPU                                               *
 *   Outputs 100-107 (R10) = CPU_TO_ABM                                         *
 *   Internal Coils 160-165 = FLAGS                                             *
 *   Timer 600 = TIMER_1                                                        *
 *                                                                              *
 * ******************************************************************************


+[   Start of Program  ]-


 << RUNG   1 >>

 * ******************************************************************************
 *                                                                              *
 *                                                                              *
 *   if ABM_TO_CPU = 0x0F4                                                      *
 *                                                                              *
 ********************************************************************************

  160                                                          R 000
+--]/[-----------------------------------------------------------+[D STR1 FUN 51]
|                                                              |
|                                                              |
+                                                             +[BINBCD FUN 86]
|                                                              |
|                                                              | K 0244
+                                                             +[ CMPR FUN 70 ]
|
|
 << RUNG    2 >>

  160      773                                                          160
+--]/[-----] [-------------------------------------------------------------(SET)
|
```

```
02/14/89     12:06:32     GE FANUC LOGICMASTER 1f documentation    Page   2
                              TRANSFER TO/FROM ABM
                              SERIES ONE PLUS CPU


<< RUNG    3 >>

* ************************************************************************
*                                                                        *
*                                                                        *
*   WORK_REG = TIMER_1;                                                  *
*                                                                        *
************************************************************************

 160     161                                                  R 600
+--] [-----]/[-----------------------------------------------+[D STR0 FUN 50]
|                                                           |
|                                                           | R 500
+                                                          +[D OUT0 FUN 60]
|                                                           |
|                                                           |          161
+                                                          +---------(SET)
|

<< RUNG    4 >>

* ************************************************************************
*                                                                        *
*                                                                        *
*   CPU_TO_ABM = WORK_REG(LSB);                                          *
*                                                                        *
************************************************************************

 161     165                                                  R 500
+--] [-----]/[-----------------------------------------------+[D STR1 FUN 51]
|                                                           |
|                                                           | R 010
+                                                          +[D OUT1 FUN 61]
|                                                           |
|                                                           |          165
+                                                          +---------(SET)
|
```

```
02/14/89     12:06:36     GE FANUC LOGICMASTER 1f documentation    Page   3
                              TRANSFER TO/FROM ABM
                              SERIES ONE PLUS CPU


|
|  << RUNG    5 >>
|
|  * ************************************************************************
|  *                                                                        *
|  *                                                                        *
|  *   if ABM_TO_CPU = 0x0F5                                                *
|  *                                                                        *
|  ************************************************************************
|
|   161                                                        R 000
+--] [-----------------------------------------------------+[D STR1 FUN 51]
|                                                          |
|                                                          |
+                                                         +[BINBCD FUN 86]
|                                                          |
|                                                          | K 0245
+                                                         +[ CMPR FUN 70 ]
|
|
|  << RUNG    6 >>
|
|   161      773      162                                          162
+--] [------] [------]/[-------------------------------------------------(SET)
|
|
|  << RUNG    7 >>
|
|  * ************************************************************************
|  *                                                                        *
|  *                                                                        *
|  *   CPU_TO_ABM = WORK_REG(MSB);                                          *
|  *                                                                        *
|  ************************************************************************
|
|   162                                                        R 501
+--] [-----------------------------------------------------+[D STR1 FUN 51]
|                                                          |
|                                                          | R 010
+                                                         +[D OUT1 FUN 61]
|
|
```

```
02/14/89     12:07:44     GE FANUC LOGICMASTER 1f documentation    Page   4
                              TRANSFER TO/FROM ABM
                              SERIES ONE PLUS CPU


  << RUNG    8 >>

  * ************************************************************************
  *                                                                        *
  *                                                                        *
  *   RESET_FLAGS;                                                        *
  *                                                                        *
  ************************************************************************

   162                                                              160
+--] [-----------------------------------------------------------+---------(RST)
|                                                                |
|                                                                |          161
+                                                               +---------(RST)
|                                                                |
|                                                                |          162
+                                                               +---------(RST)
|                                                                |
|                                                                |          165
+                                                               +---------(RST)
|
|  << RUNG    9 >>
|
|  164    T 600                                                    K 9999  T 600
+--]/[-----]/[-----------------------------------------------------[PRESC]--(TMR)
|
|
|  << RUNG   10 >>
|
|
+[ENDSW}-
|
|
```

Example 2 - Sending Commands to an OIT

To send commands from the ASCII/BASIC Module to an Operator Interface Terminal (OIT), you must send escape sequences. For example, to send a command to print file number 9 requires the following line of code:

```
10 PRINT CHR(27), "[9w",
```

The comma at the end of the sequence causes the cursor to remain where it was located after the file was printed. A command to place the cursor at a specific location on an OIT screen would require the following line:

```
10 PRINT CHR(27) "[11;49H",
```

That command will cause the cursor to be located in row 11, column 49. To print a value in a certain format, such as a timer in .1 second increments, requires first printing a format, then the data, as shown below.

```
10 PRINT USING(###.#)
20 PRINT TMR(1)
30 PRINT USING(0)
```

Refer to the description of the USING command for details on setting up different formats. The PRINT USING(0) command is a reset to default conditions.

When powering-up an ASCII/BASIC Module, it may be useful to wait a specific period of time before starting execution of a particular program, since erroneous characters may be received while the connecting device is also powering-up. This delay can be achieved by using the CLOCK command as shown in the following example.

```
20 CLOCK 1
30 IF (TIME>10) THEN GOTO 50
40 GOTO 30
50 CLOCK 0
```

In line 20, the internal clock is started, and in line 30 it is checked for a duration of ten seconds. When ten seconds has passed, the clock is stopped. At this point, the program will begin execution.

Appendix A
Commands, Statements, and Operators: Reference Tables

Table A-1. Commands

COMMAND	FUNCTION	EXAMPLE(S)
RUN	Execute a program	RUN
CONT	CONTinue after a STOP or Ctrl C	CONT
LIST	LIST program to the console device	LIST LIST 10-50
NEW	Erase the program stored in RAM	NEW
NULL	Set NULL count after carriage return-line feed	NULL NULL 4
RAM	Evoke RAM mode, current program in RAM memory	RAM
ROM	Evoke ROM mode, current program in Archival Storage Area	ROM ROM 3
XFER	Transfer a program from archival storage area to RAM	XFER
BURN	Save the current program in archival storage area	BURN

Table A-2. Statements

STATEMENT	FUNCTION	EXAMPLE(S)
CLEAR	CLEAR variables, interrupts, and strings	CLEAR
CLEARI	CLEAR interrupts	CLEARI
CLEARS	CLEAR stacks	CLEARS
CLOCK1	Enable real time clock	CLOCK1
CLOCK0	Disable real time clock	CLOCK0
DATA	DATA to be read by READ statement	DATA 100
READ	READ data in DATA statement	READ A
RESTORE	RESTORE READ pointer	RESTORE
DIM	Allocate memory for arrayed variables	DIM A(20)
DO	Set up loop for WHILE or UNTIL	DO
UNTIL	Test DO loop condition (loop if false)	UNTIL A=10
WHILE	Test DO loop condition (loop if ture)	WHILE A=B
END	Terminate program execution	END
FOR-TO-{STEP}	Set up FOR-NEXT loop	FOR A=1 TO 5
NEXT	Test FOR-NEXT loop condition	NEXT A
GOSUB	Execute subroutine	GOSUB 1000
RETURN	RETURN from subroutine	RETURN
GOTO	GOTO program line number	GOTO 500
ON GOSUB	Conditional GOSUB	ON A GOSUB 2, 6
ON GOTO	Conditional GOTO	ON A GOTO 5, 20
IF-THEN-ELSE	Conditional test	IF A<B THEN A=0
INPUT	INPUT a string or variable	INPUT A
LET	Assign a variable or string a value (LET is optional)	LET A=10
ONERR	ONERR or GOTO line number	ONERR 1000
ONTIME	Generate an interrupt when TIME is equal to or greater than ONTIME argument-line number is after comma	ONTIME 10, 1000

Table A-2. Statements - Continued

STATEMENT	FUNCTION	EXAMPLE(S)
PRINT	PRINT variables, strings or literals. P. is shorthand for PRINT	PRINT A
PH0.	PRINT HEX mode with zero suppression	PH0.A
PH1.	PRINT HEX mode with no zero suppression	PH1.A
PUSH	PUSH expressions on argument stack	PUSH 10, A
POP	POP argument stack to variables	POP A, B, C
REM	REMark	REM DONE
RETI	RETurn from Interrupt	RETI
STOP	Break program execution	STOP
STRING	Allocate memory for STRINGs	STRING 50, 10
IDLE	Wait for interrupt	IDLE
RROM	Run a program in ROM	RROM 3
LD@	Load top of stack from user-specified location	LD@ 1000H LD@ A
ST@	Store top of stack at user-specified location	ST@ 1000H ST@ A

Table A-3. Dual Operand Operators

OPERATOR	FUNCTION	EXAMPLE(S)
+	Addition	1 + 1
/	Division	10/2
**	Exponentiation	2**4
*	Multiplication	4*4
-	Subtraction	8 - 4
.AND.	Logical AND	10.AND.5
.OR.	Logical OR	2.OR.1
.XOR.	Logical Exclusive OR	3.XOR.2

Table A-4. Single Operand Operators

OPERATOR	FUNCTION	EXAMPLE(S)
ABS( )	Absolute Value	ABS(-3)
NOT( )	Ones Complement	NOT(0)
INT( )	Integer	INT(3.2)
SGN( )	Sign	SGN(-5)
SQR( )	Square Root	SQR(100)
RND	Random Number	RND
LOG( )	Natural Log	LOG(10)
EXP( )	"e" (2.7182818) TO THE X	EXP(10)
SIN( )	Returns the sine of argument	SIN(3.14)
COS( )	Returns the cosine of argument	COS(0)
TAN( )	Returns the tangent of argument	TAN(.707)
ATN( )	Returns arctangent of argument	ARN(1)

Table A-5. Special Function Operators

OPERATOR	FUNCTION	EXAMPLE(S)
CBY( )	Read program memory	P.CBY(4000)
DBY( )	Read/assign internal data memory	DBY(99)=10
XBY( )	Read/assign external data memory	P.XBY(10)
GET	Read console	P.GET
TIME	Read/assign the real time clock	P.TIME

Table A-6. Stored Constants

CONSTANT	FUNCTION	EXAMPLE(S)
PI	PI = 3.1415926	PI

Table A-7. GE Fanuc - NA Custom BASIC Commands

COMMAND	FUNCTION	EXAMPLE(S)
LSTR	Calculate length of a string	LSTR $(10), HOWLONG
SWITCH	Read DIP switch settings on ASCII/BASIC Module. Allows program to monitor board settings	SWITCH SW
MATCH	Does a String comparison	MATCH $(1),"YES",FLAG
INPB	Fetches specified number of characters from the serial port and stores them in the specified string.	INPB $(2),10,2000,FLAG
RENUM@	Renumber the BASIC line numbers	RENUM 50, 10, 10
TRANSFER	Send/receive data with Series One CPU	TRANSFER W,2,1,FLAG
PORTSET	Select 300, 600, 1200, 2400, 4800, 9600, or 19200 baud	PORTSET 9600
MOVE	Sets a single bit or field of bits	MOVE VAR2,VAR1,12,4
SPLIT	Reads a single bit or field of bits	SPLIT VAR1,VAR2,9,3
DELPRM	Deletes a program stored in the archival storage area	DELPRM 8
LED0 1 or 2	Turns the specified LED off	LED0 1
LED1 1 or 2	Turns the specified LED on	LED1 2

Appendix B
Miscellaneous Programming Information

Floating Point Format

MCS BASIC-52 stores all floating point numbers in a normalized packed BCD format with an offset binary exponent. The simplest way to demonstrate the floating point format is to use an example. If the number PI (3.1415926) were stored in location X, the following would appear in memory:

Table B-1. Sample Floating Point Format

LOCATION	VALUE	DESCRIPTION
X	81H	Exponent - 81H = 10**1, 82H = 10**2, 80H = 10**0, 7FH = 10** -1 etc. The number zero is represented with a zero exponent.
X-1	00H	Sign Bit - 00H = Positive, O1H = Negative. Other bits are used as temps only during a calculation.
X-2	26H	Least significant two digits.
X-3	59H	Next least significant two digits.
X-4	41H	Next most significant two digits.
X-5	31H	Most significant two digits.

Because MCS BASIC-52 normalizes all numbers, the most significant digit is never a zero unless the number is zero.

Storage Allocation

Two 16-bit pointers stored in external memory control the allocation of strings and variables. An additional two pointers control the allocation of scalar variables and dimensioned variables. These pointers are located and defined as follows:

Table B-2. Location and Definition of Pointers

LOCATION (H-L)	NAME	DESCRIPTION
10AH - 10BH	MTOP	The top of RAM that is assigned to BASIC.
104H - 105H	VARTOP	VARTOP = MTOP - (the number of bytes of memory that you have allocated for strings). If strings are not used, VARTOP = MTOP.
106H - 107H	VARUSE	After a NEW, CLEAR, or RUN is executed, VARUSE = VARTOP. Every time you assign or use a variable, VARUSE is decremented by a count of 8.
108H - 109H	DIMUSE	After a NEW, CLEAR, or RUN is executed, DIMUSE = [the length of the user program that is in RAM memory + the starting address of the user program in RAM (512) + the length of one floating point number (6)]. If no program is in RAM memory, DIMUSE = 518 after a CLEAR is executed.

MCS BASIC-52 stores string variables between VARTOP and MTOP. $(0) is stored from VARTOP to VARTOP + (user defined string length + 1), $(1) is stored from VARTOP + (user defined string length

+ 1) + 1 to VARTOP + 2 * (user defined string length + 1), etc. If MCS BASIC-52 attempts to access a string that is outside the bounds established by MTOP, a MEMORY ALLOCATION error is generated.

Now, scalar variables are stored from VARTOP "down" and dimensioned variables are stored from DIMUSE "up". When you dimension a variable either implicitly or explicitly, the value of DIMUSE increases by the number of bytes required to store that dimensioned variable. For example, if you execute a DIM A(10) statement, DIMUSE would increase by 66. This is because you are requesting storage for 11 numbers (A(0) through A(10)) and each number required 6 bytes for storage and 6 * 11 = 66.

As mentioned in the previous example, every time you define a new variable the VARUSE pointer decrements by a count of 8. Six of the eight counts are due to the memory required to store a floating point number, and the other two counts are the storage required for the variable name (i.e., A1, B7, etc.). The variable B7 would be stored as follows:

Table B-3. Storage Allocation

LOCATION	VALUE	DESCRIPTION
X	37H	The ASCII value 7. If B7 were a dimensioned variable, the most significant bit of this location would be set. In Version 1.1, this location always contains the ASCII value for the last character used to define a variable.
X-1	42H	This location contains the ASCII value of the first character used to define a variable plus 26 * the number of characters used to define a variable, if the variable contains more than 2 characters.
X-2 thru X-7	??	The next six locations would contain the floating point number that the variable is assigned to, if the variable were a scalar variable. If the variable were dimensioned, X-2 would contain the limit of the dimension (i.e., the maximum number of elements in the array) and X-3:X-4 would contain the base address of the array. This address is equal to the old value of the DIMUSE pointer before the array was created.

Whenever a new scalar or dimensioned variable is used in a program, MCS BASIC-52 checks both the DIMUSE and VARUSE pointers to make sure that VARUSE > DIMUSE. If the relationship is not true, a MEMORY ALLOCATION error is generated.

In Summary:

Strings are stored from VARTOP to MTOP.

Scalar variables are stored from VARTOP "down", and VARUSE points to the next available scalar location.

Dimensioned variables are stored from the end of the user program in RAM "up". If no program is in RAM, this location is 518. DIMUSE keeps track of the number of bytes you have allocated for dimensioned variables.

If DIMUSE >= VARUSE, a MEMORY ALLOCATION error is generated.

GFK-0249

Format of an MCS BASIC-52 Program

Line Format

Each line of MCS BASIC-52 text consists of tokens and ASCII characters, plus four bytes of overhead. Three of these four bytes are stored at the beginning of every line. The first byte contains the length of a line in binary, and the second two bytes are the line number in binary. The fourth byte is stored at the end of the line, and this byte is always a 0DH or a carriage return in ASCII. An example of a typical line is shown below. You should assume that this is the first line of a program in RAM.

```
10   FOR I = 1 TO 10 : PRINT I : NEXT I
```

Table B-4. Line Format

LOCATION	BYTE	DESCRIPTION
512	11H	The length of the line in binary (17D bytes)
513	00H	High byte of the line number.
514	0AH	Low byte of the line number.
515	0A0H	The token for "FOR".
516	49H	The ASCII character "I".
517	0EAH	The token for "=".
518	31H	The ASCII character "1".
519	0A6H	The token for "TO".
520	31H	The ASCII character "1".
521	30H	The ASCII character "0".
522	3AH	The ASCII character ":".
523	89H	The ASCII character "PRINT".
524	49H	The ASCII character "I".
525	3AH	The ASCII character ":".
526	97H	The token for "NEXT".
527	49H	The ASCII character for "I".
528	0DH	End of line (carriage return).

To find the location of the next line, the length of the line is added to the location where the length of the line is stored. In this example, 512 + 17D = 529, which is where the next line is stored.

The END of a program is designated by the value 01H. So, in the previous example, if line 10 were the only line in the program, location 529 would contain the value 01H. A program simply consists of a number of lines packed together in one continuous block with the last line ending in a 0DH, 01H sequence.

Archival Storage Area Format

The format of the Archival Storage Area consists of the same line and program format previously described, except that each program in the Archival Storage Area begins with the value 55H. The value 55H is only used by MCS BASIC-52 to determine if a valid program is present. If you type ROM 6, MCS BASIC-52 actually goes through the first program stored in the Archival Storage Area line by line until the END OF PROGRAM (01H) is found. Then, it examines the next location to see if a 55H is stored in that location. It then goes through that program line by line. This process is repeated 6 times.

If the character 55H is not found after the end of a program, MCS BASIC-52 will return with the PROM MODE error message. This would mean that less than six programs were stored in the Archival Storage Area.

The first program stored in the Archival Storage Area always begins at location 8010H, and this location will always contain a 55H. The actual user program, however, will begin at location 8011H.

Appendix C
Other Software Packages

Using a Software Package Other Than ABMHelper

The following examples show how VTERM, a commercially available terminal emulator package, can be used for transferring files between an ASCII/BASIC Module and a diskette on a computer. *VTERM is a product of Coefficient Systems Corporation, 611 Broadway, New York, New York 10012.*

You must first, have VTERM installed and running as a terminal emulator. Use the sample screen below as a general guide. Details of the setup depend on the specific hardware configuration, desired baud rate, and other criteria.

```
S E T U P    1 - Terminal, Communications and Printer    Setup Name=VTERM

 TERMINAL    Type: VT100 (ANSI)   COMMUNICATIONS Rate: 9600
         New line: OFF               Data bits/parity: 8 NONE
      Wrap around: ON                Receive parity: CHECK
           Scroll: NORMAL                 Stop bits: ONE
           Screen: NORMAL VIDEO        Auto xon/xoff: ON
      Cursor type: UNDERLINE             Local echo: OFF
      Margin bell: OFF                     Com port: 2
      DA response: ADVANCED VIDEO       On line/local: ON LINE
      Status line: ON
  Backspace sends: DELETE
 Shift-3 displays: #
    Screen format: 80 x 24         PRINTER   Output to: LPT1:
132 column method: SCROLLING              After PrtSc: NO FORMFEED
  Window movement: MANUAL                      Extent: ENTIRE SCREEN
Scrollback buffer: ALLOW OVERFLOW              Wide: OFF

       T       T       T       T       T       T       T       T       T
234567891123456789212345678931234567894123456789512345678961234567897123456

Use Arrow keys to select, + to change.   F5=Next Setup, F6=Help, Esc=End Setup
```

When the above screen is displayed, follow the on-screen information to make selections or change parameters as required. After completing the setup in this screen, press the F5 key to select the following screen, and set the parameters as shown.

```
S E T U P    2 - File Transfer                          Setup Name=VTERM


                        Protocol: ASCII TEXT


                 Maximum send rate: AT BAUD RATE
                  Wait after lines: FOR SINGLE CHAR
                        Which char: (62)
                       End of line: CR-LF
      Convert null lines to a space: NO
            Remove escape sequences: NO
                Stop upon receiving:



Use Arrow keys to select, + to change.   F5=Next Setup, F6=Help, Esc=End Setup
```

Transfer a Program from ASCII/BASIC Module Memory to Diskette

One of the features of this terminal emulator is that it allows you to transfer a program developed and stored on the ASCII/BASIC Module to disk storage on a computer. The procedure required to transfer a program stored in the ASCII/BASIC Module's memory to a diskette in a computer is described below.

- Point the ASCII/BASIC Module to the program to be saved using the ROM1, ROM2, RAM, or similar commands.
- Type LIST. Do not press the Return or Enter key.
- Type Alt R, which will cause the setup screen shown below to be displayed.
- Type in the desired filename, then set the parameters as shown.

```
Filename : FILENAME
Protocol : ASCII TEXT

If file exists : PROMPT
Ctrl-z Marks End : NO
Remove Characters : (press Ctrl/END with the cursor here)
```

- Type Alt R again.
- Press the Return key. At this point, the program will begin transferring from ASCII/BASIC Module memory to the computer.
- When the program has finished transferring, press Alt A (to abort the transfer operation).
- Your file should now be stored on disk.

Retrieve a Program on Diskette and Store to the ASCII/BASIC Module

To retrieve a program stored on diskette, and load it into the ASCII/BASIC Module's memory, use the following procedure:

- Type RAM and press Return. On many Personal Computers, this will be the Enter key.
- Type NEW, then press Return (or Enter). Ignore this step if you want to merge in a program.
- Type Alt S, you will get the setup screen that was shown above. Enter the correct filename, and make sure the parameters are the same as shown on the previous screen.
- Type Alt S, this will initiate the disk to ASCII/BASIC Module file transfer.
- The file transfer may end with a *SYNTAX ERROR* message from the ASCII/BASIC Module.
- Press Return or (Enter), as applicable for your computer.
- Wait until the RXD and TXD LEDs on the ASCII/BASIC Module stop blinking.
- The requested file has been transferred to the ASCII/BASIC Module's memory and is now ready to run.

Appendix D
ASCII Code List

Table D-1. ASCII Code List

ASCII Character	Decimal Value	Hexadecimal Value	ASCII Character	Decimal Value	Hexadecimal Value	ASCII Character	Decimal Value	Hexadecimal Value
NUL	0	00	0	48	30	[	91	5B
SOH	1	01	1	49	31	\	92	5C
STX	2	02	2	50	32	]	93	5D
ETX	3	03	3	51	33	^	94	5E
EOT	4	04	4	52	34	-	95	5F
ENQ	5	05	5	53	35	'	96	60
ACK	6	06	6	54	36	a	97	61
BEL	7	07	7	55	37	b	98	62
BS	8	08	8	56	38	c	99	63
HT	9	09	9	57	39	d	100	64
LF	10	0A	:	58	3A	e	101	65
VT	11	0B	;	59	3B	f	102	66
FF	12	0C	<	60	3C	g	103	67
CR	13	0D	=	61	3D	h	104	68
SO	14	0E	>	62	3E	i	105	69
SI	15	0F	?	63	3F	j	106	6A
DLE	16	10	@	64	40	k	107	6B
DC1	17	11	A	65	41	l	108	6C
DC2	18	12	B	66	42	m	109	6D
DC3	19	13	C	67	43	n	110	6E
DC4	20	14	D	68	44	o	111	6F
NAK	21	15	E	69	45	p	112	70
SYN	22	16	F	70	46	q	113	71
ETB	23	17	G	71	47	r	114	72
CAN	24	18	H	72	48	s	115	73
EM	25	19	I	73	49	t	116	74
SUB	26	1A	J	74	4A	u	117	75
ESC	27	1B	K	75	4B	v	118	76
FS	28	1C	L	76	4C	w	119	77
GS	29	1D	M	77	4D	x	120	78
RS	30	1E	N	78	4E	y	121	79
US	31	1F	O	79	4F	z	122	7A
SP	32	20	P	80	50	{	123	7B
!	33	21	Q	81	51	/	124	7C
"	34	22	R	82	52	}	125	7D
#	35	23	S	83	53	~	126	7E
$	36	24	T	84	54	DEL	127	7F
%	37	25	U	85	55			
&	38	26	V	86	56			
'	39	27	W	87	57			
(	40	28	X	88	58			
)	41	29	Y	89	59			
•	42	2A	Z	90	5A			
+	43	2B						
,	44	2C						
-	45	2D						
.	46	2E						
/	47	2F						

A

A Quick Introduction to W-ED, 5-2
ABMHelper distribution diskette, 4-2
ABMHelper Start-Up, 4-5
ABMHelper, advantages of, 3-1
ABMHelper, configuration of, 3-4
ABMHelper, features of, 4-1
ABMHelper, from floppy disk, 4-4
ABMHelper, from hard disk, 4-4
ABMLOC, 6-4
ABS([expr]), 9-3
Addition operator, 9-1
Advantages of Using ABMHelper, 3-1
AND logical operator, 9-2
Archival Storage Area file commands, 7-1
Archival Storage Area Format, B-3
ASC( ), 10-1
ASCII code list, D-1
ASCII/BASIC Module Configuration, 1-3
ASCII/BASIC Module Hardware Features, 2-1
ASCII/BASIC Module troubleshooting, 2-9
ATN([expr]), 9-6

B

BASIC commands, custom, 2-8
BASIC commands, list of, 2-7
BASIC Language, 1-3
Baud Rate Setup, 3-4
Block or Cut and Paste Operations, 5-4

C

CBY([expr]), 11-1
Changing W-ED's Configuration, 5-1
CHR( ), 10-2
CLEARI (Clear Interrupts), 8-1
CLEARS (Clear Stacks), 8-1
CLOCK0, 8-2
CLOCK1, 8-2
Code for Register Data Transfer, 13-2
Command Line and Text Area, 5-4
Command:, 6-1
  BURN(cr), 7-2
  CONT(cr), 6-1
  LIST(cr), 6-2
  NEW(cr), 6-3
  NULL [integer](cr), 6-3
  RUN(cr), 6-1
  XFER(cr), 7-1
Commands Used in Command (Monitor) Mode, 6-8
Commands Used Only in Partitioned Mode, 2-8
Commands, Statements, and Operators Not Supported, 2-9
Commands, Statements, and Operators: Reference Tables, A-1
Commands: RAM(cr) and ROM (integer) (cr), 7-1
Comments on Logical Operators .AND., .OR., and .XOR., 9-2
Communications Port Setup, 3-5
Configuring ABMHelper (Revision 1.02 and later), 3-4
Contents of the ABMHelper Distribution Diskette, 4-2
Control functions, 4-8
COS([expr]), 9-5
CR, 8-17
Creating a Backup Distribution Diskette, 4-2
Creating a Separate W-ED Editor Diskette, 4-2
Creating a Workable ABMHelper Diskette (Floppy Disk Based System), 4-3
Ctrl E edit function, 3-2
Ctrl F edit function, 3-2
Custom commands, 2-8

D

DATA, 8-3
Data Format and Range, 6-4
DBY([expr]), 11-1
DELPRM, 2-8, 6-9
Description of GE Fanuc - NA Custom Commands, 6-3
DIP Switch Configuration, 2-5

Distribution diskette, contents of, 4-2
Distribution diskette, creating, 4-2
Division operator, 9-1
Documenting BASIC programs, 3-5
Dual Operand Operators, 9-1

E

Editing, NOSOURCE mode, 3-2
Editing, Quick SOURCE mode, 3-3
Editing, SOURCE mode, 3-2
Editor Command Summary, 5-1
Error Conditions, 6-10
Error messages, 12-1
Example 1 - Transferring Data Between the ASCII/BASIC Module and the Series One/One Plus CPU, 13-1
Example 1 - Type a Text File Using Ctrl/U, 4-11
Example 2 - Sending Commands to an OIT, 13-7
Example 2 - Sorted Directory, 4-11
Example of Timeout to Detect CPU in STOP Mode, 6-5
Example of TRANSFER Command, 13-1
Example of Using Flag, 6-5
Example:, 7-2
Examples of Applications, 13-1
EXP([expr]), 9-5
Expanded Mode of Operation, 2-6
Exponentiation operator, 9-1
Extended ASCII (Graphics) Characters, 5-4

F

Features of ABMHelper, 4-1
File commands, Archival Storage Area, 7-1
FLAG, 6-5
Floating Point Format, B-1
Format of an MCS BASIC-52 Program, B-3
FREE, 11-4
Function of the ASCII/BASIC Module, 1-1
Functions Used in RUN Mode, 6-5

G

GE Fanuc - NA Custom BASIC Commands, 2-8
GE Fanuc - NA Custom Commands, Description of, 6-3
General Hints, 4-13
General Programming/Debug Hints, 3-6
General Specifications, 1-2
GET, 11-1
GOSUB, 8-8

H

Hexadecimal Numbers, 1-3, D-1
Hints for Documenting BASIC Programs, 3-5
How Relational Expressions Work, 9-6

I

I/O reference ranges, for addressing modes, 6-4
Initial power-up message, 2-8
INPB, 2-8, 6-8
Installing W-ED and ABMHelper on a Hard Disk System, 4-3
INT([expr]), 9-3
Introduction to W-ED, 5-2

K

Keyboard Conversion Chart, 5-1
Keywords, use of, 2-7

L

LD@, 8-24
LED0 1 or 2, 2-8
LED1 1 or 2, 2-8
LEN, 11-4
Line Format, B-3
Lithium battery, replacement of, 2-10
LOG([expr]), 9-4

LSTR, 2-8, 6-6

M

Main menu screen, 4-6
MATCH, 2-8, 6-7
Memory Usage For Program Development, 2-6
Menu Option 1 - Terminal Mode, 4-6
Menu Option 2 - Save Program to Disk, 4-9
Menu Option 3 - Load a Program from Disk, 4-9
Menu Option 4 - Merge a Disk File Into Program, 4-9
Menu Option 5 - Delete a Range of Lines, 4-9
Menu Option 6 - Directory of Disk Files, 4-9
Menu Option 7 - Temporary Exit to DOS, 4-10
Menu Option 8 - Set/View Time and Date, 4-10
Menu Option 9 - Finished With ABMHelper, 4-10
Messages, error, 12-1
Messages, W-ED, 5-6
Miscellaneous Programming Information, B-1
Modes of Operation for Memory Use, 1-3
Modifying ABMHelper for use with Fast Computers, 4-11
MOVE, 2-8, 6-6
MTOP, 11-3
MTOP, use of, 1-3
Multiplication operator, 9-1

N

NOSOURCE Mode of Operation, 3-2
NOT([expr]), 9-3

O

OIT, example of sending commands to, 13-7
Operation, NOSOURCE mode, 3-2
Operation, SOURCE mode, 3-2
Operators and expressions, 9-1
Operators, precedence of, 9-6
Operators, special function, 11-1
Operators, string, 10-1
OR logical operator, 9-2
Other software packages, C-1
Other W-ED Key Actions, 5-3

P

Partitioned Mode of Operation, 2-6
Personal computers, systems tested with ABMHelper, 4-1
PI, 9-4
PORTSET, 2-8
Power-up message, 2-8
Precedence of Operators, 9-6
Printer Paging Control, 5-5
Programming hints, general, 3-6
Programming information, miscellaneous, B-1

Q

Quick SOURCE Mode Editing, 3-3

R

RAM, 7-1
READ, 8-3
Reference ranges, 6-4
Register data transfer, example of code, 13-2
Register data transfer, ladder logic example, 13-3
Relational expressions, use of, 9-6
Relay Ladder Logic for Register Data Transfer, 13-3
RENUM@, 2-8, 6-8
Replacing the Lithium Battery, 2-10
RESTORE, 8-3
Retrieve a Program on Diskette and Store to the ASCII/BASIC Module, C-2
RETURN, 8-8
RND, 9-4
ROM, 7-1
RS-232 Serial Communications Port, 2-2
Run/Trap Mode, 2-6

Running ABMHelper from a Hard Disk, 4-4
Running From Floppy Disk, 4-4

S

SGN([expr]), 9-3
Sign On Message, 2-8
SIN([expr]), 9-5
Single Operand Operators - General Purpose, 9-3
Single Operand Operators - Log Functions, 9-4
Single Operand Operators - Trig Functions, 9-5
Software packages, other, C-1
SOURCE Mode of Operation, 3-2
SOURCE/NOSOURCE Configuration, 3-1
SOURCE/NOSOURCE Sub Modes, 3-3
SPC([expr]), 8-16
Special function operators, 11-1
Special Print Formatting Statements, 8-16
Specifications, General, 1-2
SPLIT, 2-8, 6-6
SQR([expr]), 9-4
ST@, 8-24
Start-up, ABMHelper, 4-5
Starting W-ED From DOS, 5-2
Statement:, 8-1
  CLEAR, 8-1
  DIM, 8-4
  END, 8-6
  GOTO [ln num], 8-9
  IDLE, 8-22
  IF - THEN - ELSE, 8-10
  INPUT, 8-11
  LET, 8-13
  ONERR [ln num], 8-14
  ONTIME [expr],[ln num], 8-14
  POP [var], 8-19
  PRINT or P., 8-16
  PUSH [expr], 8-18
  REM, 8-20
  RETI, 8-20
  RROM (integer), 8-22
  STOP, 8-21
  STRING [expr],[expr], 8-21
Statements:, 8-1
  CLEARI and CLEARS, 8-1
  CLOCK1 and CLOCK0, 8-2
  DATA - READ - RESTORE, 8-3
  DO - UNTIL [rel expr], 8-5
  DO - WHILE [rel expr], 8-6
  FOR - TO -{STEP} - NEXT, 8-7
  GOSUB [ln num] - RETURN, 8-8
  ON [expr] GOSUB [ln num],[ln num], . . . [ln num] ON [expr] GOTO [ln num],[ln num], . . . [ln num], 8-10
  PH0., PH1., 8-18
  ST@ [expr] and LD@ [expr], 8-24
Status Display, 2-1
Storage Allocation, B-1
String operators, 10-1
Subtraction operator, 9-2
SWITCH, 2-8, 6-7
System Control Values, 11-3
System Requirements, 4-1

T

TAB([expr]), 8-16
Tabbing and Margins, 5-4
TAN([expr]), 9-5
Text Manipulation, 5-5
TIME, 11-2
Time and Date Setup, 3-5
TRANSFER, 2-8
Transfer a Program from ASCII/BASIC Module Memory to Diskette, C-2
Transfer command, example of use, 13-2
Transfer command, how to use, 13-1
Transfer Command: TRANSFER R/W, FORMAT, ABMLOC, FLAG, 6-4
Transferring Register Data, 13-2
Troubleshooting ABMHelper, 4-12
Troubleshooting Guide, 2-9
Troubleshooting the ASCII/BASIC Module, 2-9

U

Use of Keywords, 2-7

Using a Software Package Other Than
ABMHelper, C-1
Using ABMHelper, 4-4
Using the F1 - F10 Function Keys, 4-8
Using the Transfer Command, 13-1
Using User-Supplied Routines with
ABMHelper, 4-10
USING(special characters), 8-17

V

VTERM, how to use, C-1

W

W-ED, 5-2
default settings, 5-2
Features, 5-3
Function Keys, 5-3
Messages, 5-6
starting from DOS, 5-2
Workmaster Keyboard, 5-1

X

XBY([expr]), 11-1
XOR logical Exclusive OR operator, 9-2
XTAL, 11-3

GE Fanuc Automation North America, Inc., Charlottesville, Virginia