



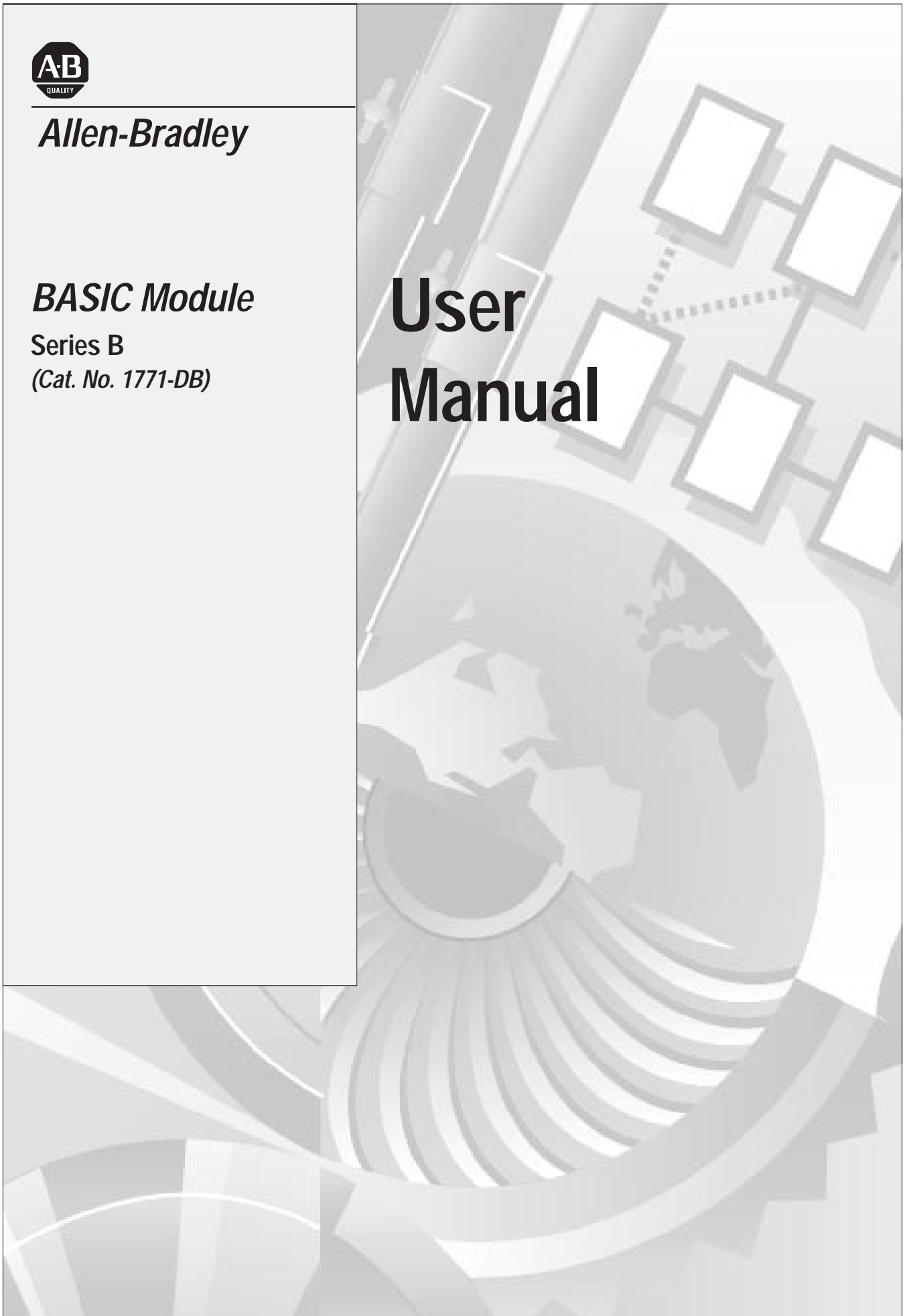
Allen-Bradley

BASIC Module

Series B

(Cat. No. 1771-DB)

User Manual



Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *General Information Safety Guidelines for Solid-State Control* (available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electro-mechanical devices that should be taken into consideration when applying products such as those described in this publication.

Reproduction of the contents of this copyrighted publication, in whole or in part, without written permission of Allen-Bradley Company, Inc., is prohibited.

Throughout this manual we use notes to make you aware of safety considerations:



ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss.

Attention statements help you to:

- identify a hazard
- avoid the hazard
- recognize the consequences

Important: Identifies information that is critical for successful application and understanding of the product.

Summary of Changes

What's in This Preface?

Read this preface if you are replacing a 1771-DB, Series A module with a 1771-DB, Series B module or are using the BASIC module for the first time. This preface discusses:

- compatibility of the 1771-DB, Series B BASIC module with the 1771-DB, Series A and with the 1746-BAS BASIC modules
- changes to this manual since the last printing

1771-DB Series A and Series B Compatibility

The Series B BASIC module is fully compatible with the Series A BASIC module. You can run the same BASIC program you ran on the Series A on the Series B module. In addition the Series B has these features.

Series B Features	Description	See
EEPROM programming support	two sockets for memory modules: one for EEPROM and the other for EPROM	Chapter 3
Additional RAM memory	24K bytes of RAM (an increase from 13K bytes)	
Five new troubleshooting LEDs	LEDs now indicate when PRT1 and PRT2 are either transmitting or receiving	Appendix C
Two fully functional serial ports	ports PRT1 and PRT2 can be configured independently for RS-232, RS-422, or RS-485; on PRT2, you do not need to jumper pins 4 and 5 together to print	Chapter 2
Error-trapping support	ONERR statement traps overflow, underflow and divide-by-zero errors expanded error support for non-hardware errors undefined in the ONERR statement	Page 11–23 CALL 38, page 12–36
DH-485 network support	DH-485 port that you can use as a network port or a programming port	Chapter 2
DF1 protocol support for remote communication	configure port PRT2 for DF1 protocol in either full- or half-duplex modes; with DF1 protocol you can communicate with external devices using a phone line, radio link, or dial-up modem	Chapter 2
Operate in 8-point or 16-point modes	in 8-point mode, the module uses 8 bits in both the input and output image tables for block transfer in 16-point mode the module also allows you to examine bits 10 – 17 for communication port status	Chapters 1 and 5
Turbo speed for faster program execution	the module operates up to four times faster run the module at the same speed as Series A for slower applications	
Reset switch support	hard reset switch initiates a full reset	
Enhanced serial port statements and modifiers	enhanced serial port statements (EOF, INPS, INPL) and port statement modifiers (# and @) aid developing programs for serial communications	Chapter 9
PLC-5 [®] floating point number support	supports PLC-5 floating point numbers; Series A did not	Chapter 8

EEPROM Programming Support

The Series B module has two sockets for memory modules: one for EEPROM and the other for EPROM. Program and erase EEPROM memory modules with the Series B module. You no longer have to use CALLs 8 and 9 to burn your EEPROM.

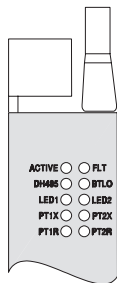
Also, you can read and run the EPROM memory module (8K, 16K, and 32K) you programmed with your Series A module on the Series B. However, you cannot program an EPROM on the Series B.



See Chapter 3 for more information.

Additional RAM Memory

The Series B module has increased RAM to 24K bytes; Series A had 13K.



Five New LED Indicators

The Series B module has increased the number of troubleshooting LEDs from 5 to 10. The LEDs now indicate when PRT1 and PRT2 are either transmitting or receiving. The ACTIVE light now flashes when you are in Command mode and remains on when you are in Run mode. Plus, you now have two user-defined LED indicators.

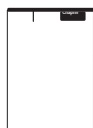


See Appendix C for more information.

Two Fully Functional Serial Ports

Ports PRT1 and PRT2 can be configured independently for RS-232, RS-422, or RS-485. For Series B, you do not need to jumper pins 4 and 5 together on PRT2 for printing.

Software handshaking is enabled as a default on Series B, but not Series A.



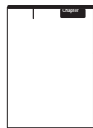
See Chapter 2 for more information.

Error-Trapping Support

The ONERR statement (page 11 -23) traps overflow, underflow, and divide-by-zero errors in the Series B module. Also, the Series B offers expanded error (CALL 38, page 12 -36) support for non-hardware errors undefined in the ONERR statement.

DH-485 Network Support

The Series B module has a DH485 port that you can use as a network port or a programming port.



See Chapter 2 for more information.

DF1 Protocol Support for Remote Communication

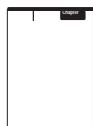
You can configure port PRT2 for DF1 protocol in either full- or half-duplex mode. With the DF1 protocol you can communicate with external devices using, for example, a phone line, radio link, or dial-up modem.



See Chapter 2 for more information.

BASIC Module Operates in 8-Point or 16-Point Mode

The BASIC module can operate in 8-point or 16-point backplane modes. In 8-point mode, the BASIC module uses 8 bits in both the input and output image tables for block transfer. In 16-point mode the BASIC module also allows you to examine bits 10 – 17 for communication port status.

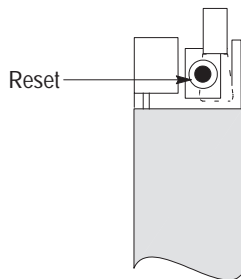


See Chapters 1 and 5 for more information.

Turbo Speed Allows Faster Program Execution

The BASIC module operates up to four times faster than before. With a C toolkit and C compiler available from one of our Pyramid Solutions Program partners, you can run C programs on the BASIC module even faster. See your Allen-Bradley representative for more details on the C toolkit. You can also run the BASIC module at the same speed as the Series A module for applications that cannot be run at a faster speed.

With the BASIC Development Software (1747-PBASE) you can shorten program development time. This powerful programming tool provides a high level BASIC programming language, powerful debugger, ASCII terminal emulator, and a thorough Help system to streamline BASIC module programming and troubleshooting.

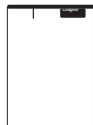


Reset Switch Support

The BASIC module has a hard reset switch located behind the module ejector tab. When this switch is pressed, the BASIC module initiates a full reset. The BASIC module reacts to this reset the same as it does when you turn on power to your I/O chassis backplane.

Enhanced Serial Port Statements and Modifiers

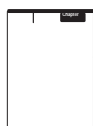
With the BASIC module's new enhanced serial port statements (EOF, INPS, INPL) and port statement modifiers (# and @) developing programs for serial communications is easier. When you use the @ operator you direct communications to port PRT1. When you use the # statement modifier you direct communications to port PRT2.



See Chapter 9 for more information.

PLC-5[®] Floating Point Number Support

The Series B module supports PLC-5 floating point numbers; Series A did not.



See Chapter 8 for more information.

Call Routine Changes and Additions

These calls are **new** to the Series B, BASIC module:

Statement	Page
CALL 0 reset the module	12 -2
CALL 14 SLC 16-bit signed integer to BASIC floating point	12 -8
CALL 15 SLC 16-bit unsigned integer to BASIC floating point	12 -9
CALL 16 enable/disable DF1 packet interrupt	12 -10
CALL 18 re-enable control C break function	12 -11
CALL 19 disable the control C break function	12 -12
CALL 24 BASIC floating point to SLC 16-bit signed integer	12 -15
CALL 25 BASIC floating point to SLC 16-bit binary	12 -16
CALL 29 read/write to PLC/SLC from module internal string	12 -18
CALL 49 read remote DH-485 SLC data file	12 -44
CALL 50 write to remote DH-485 SLC data file	12 -50
CALL 83 display DH485 port setup	13 -11
CALL 84 transfer DH-485 CIF to BASIC input buffer	13 -12
CALL 85 transfer BASIC output buffer to DH-485 CIF file	13 -13
CALL 86 check DH-485 interface remote write status	13 -14
CALL 87 check DH-485 interface file remote read status	13 -15
CALL 88 BASIC floating point to PLC-5 floating point	13 -16
CALL 89 PLC-5 floating point to BASIC floating point	13 -17
CALL 90 read remote DH-485 data file to BASIC input buffer	13 -18
CALL 91 write BASIC output buffer to remote DH-485 data file	13 -22
CALL 92 read remote DH-485 CIF to BASIC input buffer	13 -26
CALL 93 write output buffer to remote DH-485 CIF file	13 -29
CALL 94 display current PRT1 port setup	13 -32
CALL 95 get number of characters in PRT1 buffers	13 -32
CALL 96 clear PRT1 input/output buffers	13 -33
CALL 97 enable PRT2 DTR	13 -33
CALL 98 disable PRT2 DTR	13 -34
CALL 100 download/program assembly language to EEPROM	13 -35
CALL 101 upload user (E)EPROM code to host	13 -35
CALL 103 print PRT1 output buffer and pointer	13 -36
CALL 104 print PRT1 input buffer and pointer	13 -37
CALL 105 reset PRT1 to default settings	13 -37
CALL 108 enable DF1 driver communications	13 -38
CALL 112 user LED control	13 -47
CALL 113 disable DF1 driver communications	13 -47
CALL 114 transmit DF1 packet	13 -48
CALL 115 check DF1 status	13 -49
CALL 116 call user defined assembly language routine	13 -50
CALL 117 get DF1 packet length	13 -51
CALL 118 PLC/SLC unsolicited writes	13 -52
CALL 120 clear BASIC module input and output buffers	13 -57
CALL 122 read remote DF1 PLC data file	13 -58
CALL 123 write to remote DF1 PLC data file	13 -66

The definitions of these calls have *changed*:

Important: The Series A definitions are not supported in the Series B.

CALL	Series A definition	Series B definition	Page
30	peripheral port support parameter set: enables pins 6 and 8 pins 4 and 5 are always enabled	PRT2 port support parameter set: enables pins 4, 5, 6 and 8	12 -20
32	save program to data recorder ^①	enable/disable processor interrupt	12 -22
33	verify program with data recorder ^①	transfer data from PRT1 or PRT2 to CPU files	12 -23
34	load program from data recorder ^①	transfer data from CPU files to PRT1 or PRT2	12 -29
38	save labeled program to data recorder (1770-SB only) ^①	expanded ONERR	12 -36
39	load labeled program from data recorder (1770-SB only) ^①	3.3-Digit Signed, BCD to BASIC Floating Point	12 -38

^①The Series B module does not support the 1770-SA/SB data recorder (cassette recorder).

Important: Because these calls are no longer needed to program PROMs, the Series B **does not support** these Series A calls. Note that if you call one of these calls you do not receive an error message, but return to the main program without performing a task.

CALL	Series A definition
8	disable interrupts
9	enable interrupts
79	blink the active LED by default



See Chapters 12 and 13 for more information on call routines.

1771-DB Series B and 1746-BAS Compatibility

The 1771-DB Series B, BASIC module provides 1771 backplane support instead of SLC™ backplane support. The data types, commands, statements and call routines are similar between the two modules. These calls are different:

This 1771-DB call:	Is equivalent to this 1746-BAS call:
32 (enable/disable processor interrupt)	20 and 21
33 (transfer data from PRT1 or PRT2 to CPU files)	22
34 (transfer data from BTW buffer to PRT1 or PRT2)	23
49 (read remote DH-485 SLC data file)	27
50 (write to remote DH-485 SLC data file)	28

Changes to the Manual Since the Last Printing

We have corrected these items that appeared in the previous version of this manual (1771-6.5.113; November 1994). We show these changes with revision bars:

We changed:	On page:
Software handshaking enabled as default on Series B, but not on Series A	SOC-2
Catalog number reference from 1746-PBASE to 1747-PBASE	SOC-3
Improved organization of jumper setting table	1-6
Catalog number reference from 1770-XY to 1771-XZ	3-8
Highlighted reference to character limit for greater visibility	4-6
Added space in line 5 of code	5-7
Added parentheses to syntax reference	11-20
Added line of code	12-28, 12-33, 12-56
Added I/O argument in code	12-40
Corrected example text in CALL 72	13-4

Notes:

Using This Manual

What's in This Preface?

This introduction describes how to properly and efficiently use this manual.



publication 1771-6.5.113

This introduction tells you:

- the purpose of this manual
- who should use this manual
- how to use this manual
- abbreviations and conventions
- related publications
- Allen-Bradley support

Purpose of This Manual

Use this manual as guide for the design, installation, and programming of the BASIC module (1771-DB, Series B). It describes the procedures for installing and using this module. As well, the manual provides reference information for programming the BASIC module.

Who Should Use This Manual

Use this manual if you are responsible for designing, installing, programming, or troubleshooting control systems that use the BASIC module with Allen-Bradley PLC[®] processors.

You should:

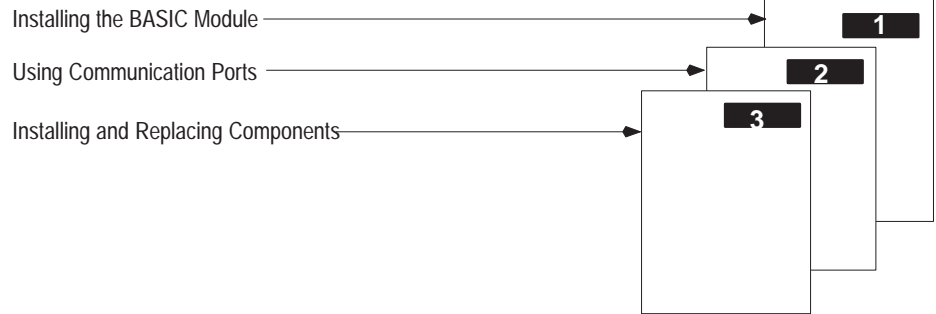
- have a basic understanding of PLC products
- understand programmable controllers
- be able to interpret the ladder logic instructions required to control your application
- be familiar with BASIC programming

Contact your local Allen-Bradley representative for information on available training courses before using this product if you are not familiar with the above items.

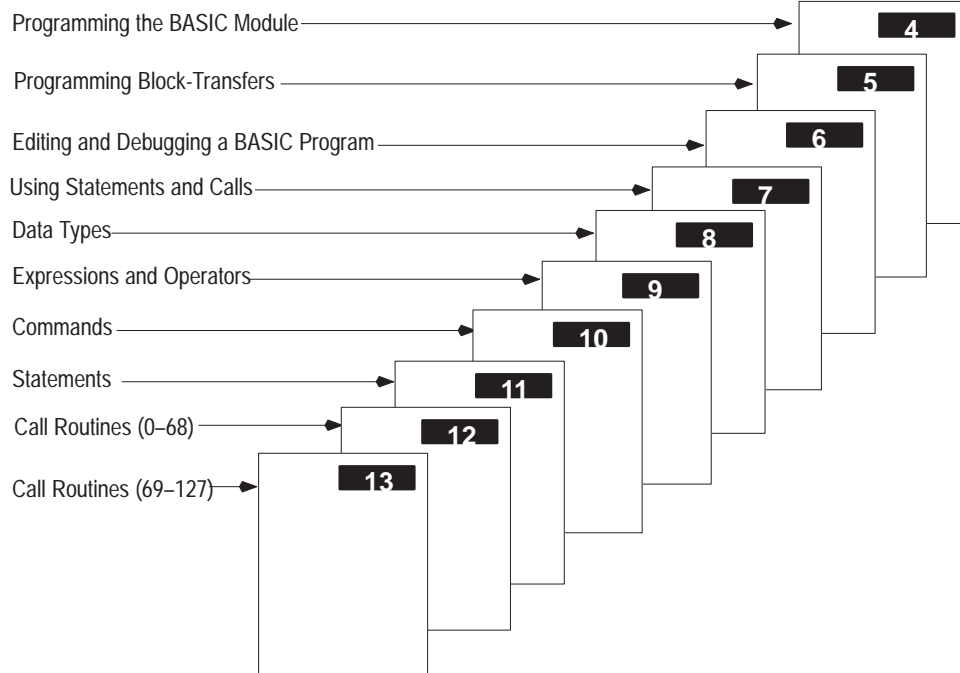
How To Use This Manual

This manual is designed so you can follow it to install your hardware and program your BASIC module.

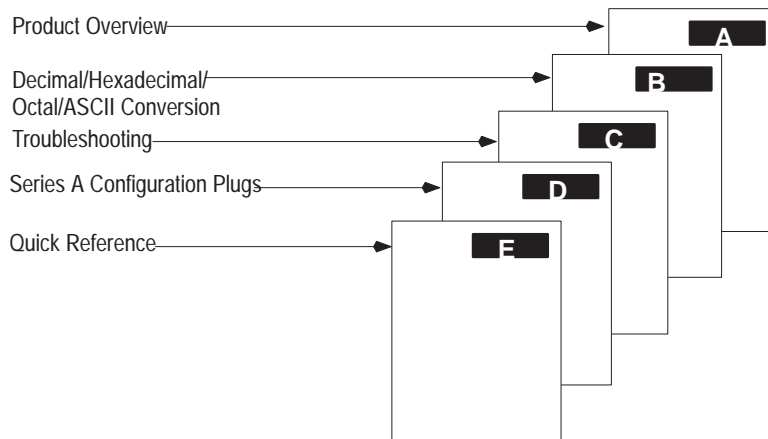
Hardware



Programming



Reference



Terms and Abbreviations







Throughout this manual, we abbreviate some terms. The terms and abbreviations listed in this table are specific to this product.

For a complete listing of Allen-Bradley terminology, refer to the Allen-Bradley Industrial Automation Glossary, (AG-7.1).

Term/Abbreviation	Definition
ASCII port	port used to connect to foreign devices. You can configure either PRT1 or PRT2 to be an ASCII port
BASIC	the BASIC-52 programming language
BASIC module	BASIC Module (catalog number 1771-DB, Series B)
BTR buffer	Block Transfer Read buffer
BTW buffer	Block Transfer Write buffer
console device	the device connected to the BASIC module program port. This device is used as an interface between the user and the BASIC program
CIF	Common Interface File
DF1 protocol	use this protocol to communicate with a node. You can configure PRT2 for DF1 protocol
dimensioned variable	a variable that includes an expression
DH-485	network communication protocol
EPROM	Erasable Programmable Read Only Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
input buffer	BASIC module input buffer (includes BTR buffer). Ports PRT1 and PRT2 also have an input (receive) buffer.
memory module	BASIC module EPROM (read only), UVPRM (read only), EEPROM, MTOP
network port	port used to connect to a DH-485 network. You can configure the DH485 port to be a network port
PBASE	BASIC Development Software (catalog number 1747-PBASE)
PLC	Programmable Logic Controller
program port	port used to program the BASIC module. You can configure either port PRT1 or port DH485 of the BASIC module as the program port.
output buffer	BASIC module output buffer (includes BTW buffer). Ports PRT1 and PRT2 also have an output (transmit) buffer.
RS-232/423	serial communication interface
RS-422	differential communication interface
RS-485	network communication interface
RAM	Random Access Memory
ROM	Read Only Memory, refers to the optional memory module memory space (EEPROM or UVPRM)
SCADA	Supervisory Control and Data Acquisition
scalar variable	a variable with a single value
SLC 500	SLC 500 fixed and modular controller
UVPRM	Ultra Violet Erasable Programmable Read Only Memory (A UVPRM is a type of EPROM.)

Conventions

We use these conventions in this manual:

In this manual, we show:	Like this:
prompts and messages	Press a function key
literal text that you type	RUN
variable text that you type	<i>filename</i>
keys that you press	
that there is more information about the topic in another chapter in this manual	
that there is more information about the topic in another manual	
helpful information	

Bulleted lists provide information, not procedural steps. Numbered lists provide sequential steps or hierarchical information.

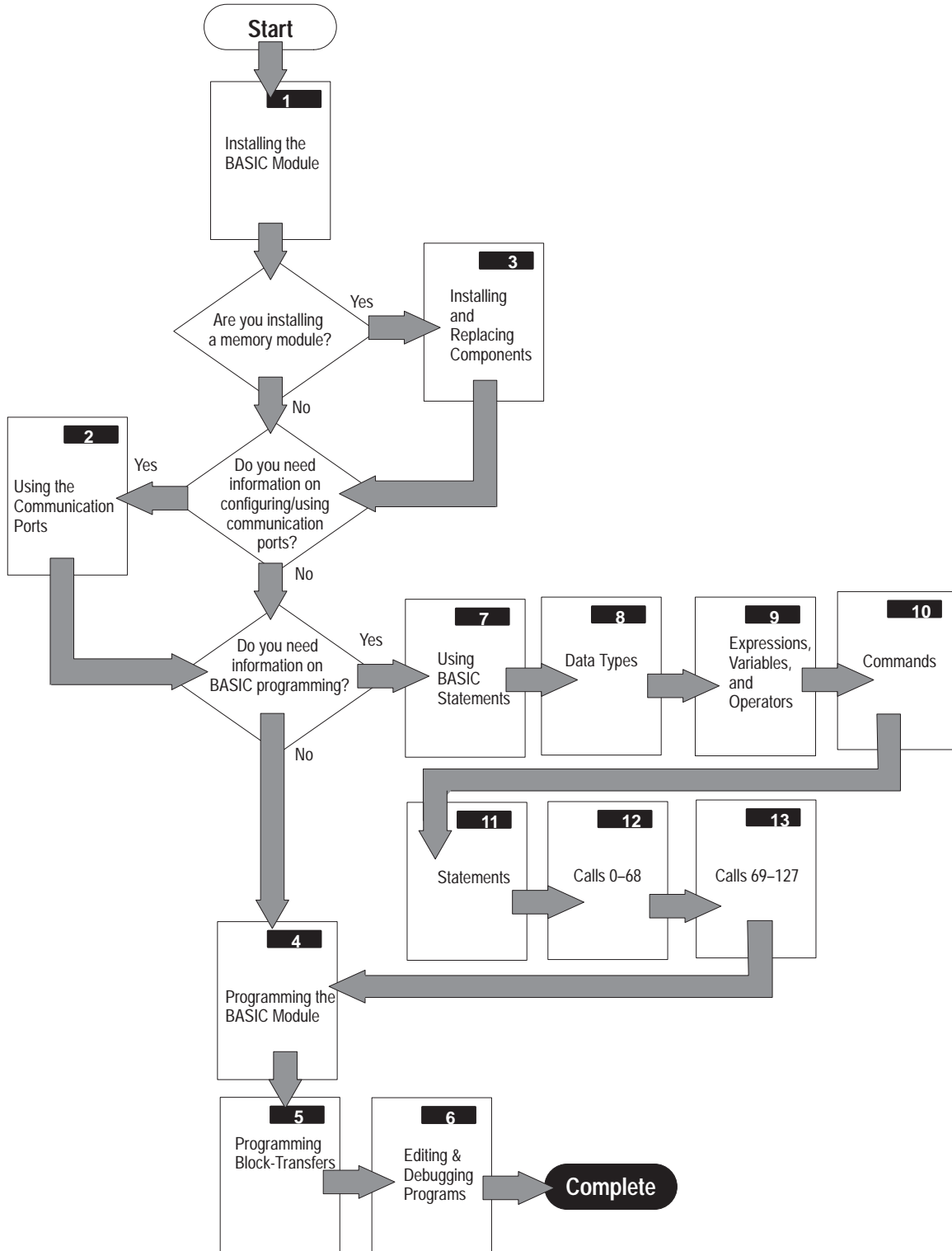
Related Publications

Publication	Publication Number
<i>BASIC Development Software Programming Manual</i>	1746-6.2
<i>Industrial Automation Wiring and Grounding Guidelines</i>	1770-4.1
<i>General Information Safety Guidelines for Solid-State Controls</i>	SGI-1.1
<i>National Electrical Code</i>	Published by the National Fire Protection Association of Boston, MA.
<i>Allen-Bradley Publication Index</i>	SD499
<i>Allen-Bradley Industrial Automation Glossary</i>	AG-7.1
<i>DH-485/RS-232C Interface Module User's Manual</i>	1747-6.12
<i>DF1 Protocol and Command Set Reference Manual</i>	1770-6.5.16

Refer to the *Allen-Bradley Publication Index* (SD499) for the appropriate programming and operations manuals for your particular PLC processor.

Getting Started

To install and program the BASIC module follow the flowchart below.



Allen-Bradley Support

Allen-Bradley offers support services worldwide, with over 75 sales/support offices, 512 authorized distributors and 260 authorized systems integrators located throughout the United States alone. As well, Allen-Bradley has representatives in every major country in the world.

Local Product Support

Contact your local Allen-Bradley representative for:

- sales and order support
- product technical training
- warranty support
- support service agreements

Technical Product Assistance

If you need to contact Allen-Bradley for technical assistance, please review the information in the appropriate chapter first. Then call your local Allen-Bradley representative.

Your Questions or Comments on this Manual

If you find a problem with this manual or have any suggestions on how we can make this manual more useful to you, please fill out and send us the enclosed Publication Problem Report.

Installing the BASIC Module

Chapter 1

What's in This Chapter? 1-1
 Guard Against Electrostatic Damage 1-1
 Unpack the Module 1-2
 Install Memory Module 1-2
 Configure Jumpers 1-3
 Determine BASIC Module Placement 1-10
 Key Backplane Connector 1-11
 Install Module in I/O Rack 1-12
 Connect Peripheral Devices 1-13
 Power up the Module 1-13
 Reset the Module 1-13
 Read the Indicator Lights 1-14
 What's Next? 1-14

Using the Communication Ports

Chapter 2

What's in This Chapter? 2-1
 Communication Ports Overview 2-1
 Communication Modes 2-2
 Handshaking 2-4
 Communication Rates 2-6
 Operating Modes 2-7
 What's Next? 2-13

Installing and Replacing Components

Chapter 3

What's in This Chapter? 3-1
 Before You Begin 3-1
 Remove the BASIC Module from the I/O Chassis 3-2
 Disassemble the BASIC Module 3-3
 Install Optional Memory Module 3-4
 Install the Battery 3-8
 Reassemble the BASIC Module 3-11
 What's Next? 3-11

Programming the BASIC Module

Chapter 4

What's in This Chapter? 4-1
 Programming Instructions 4-1
 Create a Program 4-2
 Number Program Lines 4-6
 Enter a Program 4-7
 Run and Stop a Program 4-9
 What's Next? 4-9

Programming Block-Transfers	Chapter 5	
	What's in This Chapter?	5-1
	BASIC Module Memory Organization	5-1
	Data Tables	5-2
	Block-Transfer Buffers	5-4
	Block-Transfers and the BASIC Module	5-5
	PLC-2 Family Processors Ladder Logic	5-8
	PLC-3 Family Processors Ladder Logic	5-9
	PLC-5 Family Processors Ladder Logic	5-10
	PLC-5/250 Family Processors Ladder Logic	5-12
What's Next?	5-12	
Editing and Debugging a BASIC Program	Chapter 6	
	What's in This Chapter?	6-1
	Edit a Program Line	6-1
	Delete a Program Line	6-3
	Renumber a Program	6-3
	Debug a Program	6-4
What's Next?	6-4	
Using BASIC Module Statements	Chapter 7	
	What's in This Chapter?	7-1
	Memory and Operation Calls	7-1
	Port Communication Calls	7-2
	Block-Transfer Support Calls	7-3
	Number Conversion Calls	7-4
	Clock/Calendar Calls	7-4
	String Calls	7-5
	DH-485 Communication	7-5
	DF1 Protocol Communication	7-6
	Background Operations	7-6
	Command Line Calls	7-7
	Execution Control and Interrupt Support Calls	7-7
	Input Calls	7-8
	Output Calls	7-9
	Setup Calls	7-9
	Status Calls	7-10
What's Next?	7-10	
Data Types	Chapter 8	
	What's in This Chapter?	8-1
	Argument Stack	8-1
	Control Stack	8-1
	String Data Types	8-2
	Numeric Data Types	8-3
	Backplane Conversion Data Types	8-4
	What's Next?	8-8

Expressions, Variables and Operators

Chapter 9

What's in This Chapter? 9-1
 Expressions 9-1
 Relational Expressions 9-1
 Constants 9-1
 Variables 9-2
 Order of Operations 9-3
 Arithmetic Operators 9-5
 Bitwise Operators 9-7
 Relational Operators 9-9
 Trigonometric Operators 9-10
 Functional Operators 9-11
 Logarithmic Operators 9-13
 String Operators 9-14
 Special Function Operators 9-17
 What's Next? 9-20

Commands

Chapter 10

What's in This Chapter? 10-1
 BRKPNT 10-2
 CONT 10-3
 CTRL-C 10-4
 CTRL-Q 10-5
 CTRL-S 10-6
 EDIT 10-7
 ERASE 10-8
 LIST 10-9
 NEW 10-10
 NULL 10-10
 PROG 10-11
 PROG1 10-12
 PROG2 10-13
 RAM 10-15
 REN 10-16
 ROM 10-17
 RROM 10-18
 RUN 10-19
 SNGLSTP 10-20
 VER 10-21
 XFER 10-21
 Command Line Calls 10-22
 What's Next? 10-22

Statements

Chapter 11

What's in This Chapter?	11-1
CLEAR	11-2
CLEARI	11-3
CLEARs	11-3
CLOCK0	11-4
CLOCK1	11-5
DATA	11-6
DIM	11-7
DO-UNTIL	11-8
DO-WHILE	11-9
END	11-10
FOR-TO-(STEP)-NEXT	11-11
GET	11-12
GOSUB	11-13
GOTO	11-14
IDLE	11-14
IF-THEN-ELSE	11-15
INPL	11-16
INPS	11-16
INPUT	11-17
LD@	11-18
LET	11-19
MODE	11-20
NEXT	11-21
ONDF1	11-22
ONERR	11-23
ON-GOSUB	11-24
ON-GOTO	11-25
ONTIME	11-26
PH0. and PH1.	11-27
POP	11-28
PRINT	11-29
PUSH	11-30
READ	11-31
REM	11-32
RESTORE	11-32
RETI	11-33
RETURN	11-34
ST@	11-35
STOP	11-36
STRING	11-37
What's Next?	11-38

Call Routines 0 - 68

Chapter 12

What's in This Chapter?	12-1
CALL 0: Reset Module	12-2
CALL 2: Timed Block- Transfer-Read Buffer	12-2
CALL 3: Timed Block- Transfer-Write Buffer	12-3
CALL 4: Set Block- Transfer-Write Length	12-4
CALL 5: Set Block- Transfer-Read Length	12-4
CALL 6: Block-Transfer- Write Buffer	12-5
CALL 7: Block-Transfer- Read Buffer	12-5
CALL 10: 3-Digit Signed, Fixed Decimal BCD to BASIC Floating Point	12-6
CALL 11: 16-Bit Binary to BASIC Floating Point	12-7
CALL 12: 4-Digit Signed Octal to BASIC Floating Point ...	12-7
CALL 13: 6-Digit Signed, Fixed Decimal BCD to BASIC Floating Point	12-8
CALL 14: SLC 16-Bit Signed Integer to BASIC Floating Point	12-8
CALL 15: SLC 16-Bit Unsigned Integer to BASIC Floating Point	12-9
CALL 16: Enable/Disable DF1 Packet Interrupt	12-10
CALL 17: 4-Digit BCD to BASIC Floating Point	12-11
CALL 18: Re-Enable Control C Break Function	12-11
CALL 19: Disable the Control C Break Function	12-12
CALL 20: BASIC Floating Point to 3-Digit, Signed, Fixed Decimal BCD	12-12
CALL 21: BASIC Floating Point to 16-Bit Binary	12-13
CALL 22: BASIC Floating Point to 4-Digit, Signed Octal ..	12-13
CALL 23: BASIC Floating Point to 6-Digit, Signed, Fixed BCD	12-14
CALL 24: BASIC Floating Point to SLC 16-Bit Signed Integer	12-15
CALL 25: BASIC Floating-Point to SLC 16-Bit Binary	12-16
CALL 26: BASIC Floating Point to 3.3-Digit Signed BCD ..	12-17
CALL 27: BASIC Floating Point to 4-Digit BCD	12-17
CALL 29: Read/Write to a PLC/SLC processor from the BASIC Module Internal String	12-18
CALL 30: PRT2 Port Support Parameter Set	12-20
CALL 31: Display PRT2 Port Parameters	12-21
CALL 32: Enable/Disable Processor Interrupt	12-22
CALL 33: Transfer Data from PRT1 or PRT2 to the BTR Buffer	12-23
CALL 34: Transfer Data from the BTW buffer to PRT1 or PRT2	12-29
CALL 35: Retrieve Numeric Input Character from PRT2 Port	12-34
CALL 36: Get the Number of Characters in the PRT2 Port Buffer	12-35
CALL 37: Clear the PRT2 Port Buffers	12-35
CALL 38: Expanded ONERR Restart	12-36

CALL 39:	3.3-Digit Signed, Fixed Decimal BCD to BASIC Floating Point	12-38
CALL 40:	Set the Wall Clock Time (Hour, Minute, Second) .	12-39
CALL 41:	Set Wall Clock Date (Day, Month, Year)	12-40
CALL 42:	Set Wall Clock Day of Week	12-40
CALL 43:	Retrieve Date/Time String	12-41
CALL 44:	Retrieve Date Numeric (Day, Month, Year)	12-41
CALL 45:	Retrieve Time String	12-42
CALL 46:	Retrieve Time Numeric	12-42
CALL 47:	Retrieve Day of Week String	12-43
CALL 48:	Retrieve Day of Week Numeric	12-43
CALL 49:	Read Remote DH-485 SLC Data File	12-44
CALL 50:	Write to Remote DH-485 SLC Data	12-50
CALL 52:	Retrieve Date String	12-58
CALL 60:	String Repeat	12-59
CALL 61:	String Append (Concatenation)	12-60
CALL 62:	Number to String Conversion	12-61
CALL 63:	String to Number Conversion	12-62
CALL 64:	Find a String in a String	12-63
CALL 65:	Replace a String in a String	12-64
CALL 66:	Insert String in a String	12-65
CALL 67:	Delete String from a String	12-66
CALL 68:	Determine Length of a String	12-67
	What's Next?	12-67

Call Routines 69-127

Chapter 13

	What's in This Chapter?	13-1
CALL 70:	ROM to RAM Program Transfer	13-2
CALL 71:	ROM/RAM to ROM Program Transfer	13-3
CALL 72:	RAM/ROM Return	13-4
CALL 73:	Battery-Backed RAM Disable	13-5
CALL 74:	Battery-Backed RAM Enable	13-5
CALL 77:	Protected Variable Storage	13-6
CALL 78:	Set Program Port Communication Rate	13-8
CALL 80:	Check Battery Condition	13-9
CALL 81:	User PROM Check and Description	13-10
CALL 82:	Check User Memory Module Map	13-11
CALL 83:	Display DH485 Port Parameters	13-11
CALL 84:	Transfer DH-485 Common Interface File to BASIC Input Buffer	13-12
CALL 85:	Transfer BASIC Output Buffer to DH-485 Common Interface File	13-13
CALL 86:	Check DH-485 Interface File Remote Write Status	13-14
CALL 87:	Check DH-485 Interface File Remote Read Status	13-15
CALL 88:	BASIC Floating Point to PLC-5 Floating Point . . .	13-16
CALL 89:	PLC-5 Floating Point to BASIC Floating Point . . .	13-17

CALL 90: Read Remote DH-485 Data File to
 BASIC Input Buffer 13-18

CALL 91: Write BASIC Output Buffer to
 Remote DH-485 Data File 13-22

CALL 92: Read Remote DH-485 Common Interface File to
 BASIC Input Buffer 13-26

CALL 93: Write Output Buffer to
 Remote DH-485 Common Interface File 13-29

CALL 94: Display Current PRT1 Port Setup 13-32

CALL 95: Get Number of Characters in PRT1 Buffers 13-32

CALL 96: Clear PRT1 Receive/Transmit Buffers 13-33

CALL 97: Enable Port PRT2 DTR Signal 13-33

CALL 98: Disable Port PRT2 DTR Signal 13-34

CALL 99: Reset Print Head Pointer 13-34

CALL 100: Download and Program Assembly Language Code to
 EEPROM 13-35

CALL 101: Upload User (E)EPROM Code to Host 13-35

CALL 103: Print PRT1 Transmit Buffer and Pointer 13-36

CALL 104: Print PRT1 Receive Buffer and Pointer 13-37

CALL 105: Reset PRT1 to Default Settings 13-37

CALL 108: Enable DF1 Driver Communications 13-38

CALL 109: Print the Argument Stack 13-44

CALL 110: Print the PRT2 Port Transmit Buffer and Pointer .. 13-45

CALL 111: Print the PRT2 Port Receive Buffer and Pointer .. 13-46

CALL 112: User LED Control 13-47

CALL 113: Disable DF1 Driver Communications 13-47

CALL 114: Transmit DF1 Packet 13-48

CALL 115: Check DF1 Status 13-49

CALL 116: Call User Defined Assembly Language Routine .. 13-50

CALL 117: Get DF1 Packet Length 13-51

CALL 118: PLC/SLC Unsolicited Writes 13-52

CALL 119: Reset the PRT2 Port to Default Settings 13-56

CALL 120: Clear BASIC Module I/O Buffers 13-57

CALL 122: Read Remote DF1 PLC Data File 13-58

CALL 123: Write to Remote DF1 PLC Data File 13-66

Product Overview

Appendix A

What's in This Appendix?	A-1
Features	A-1
Programming Interfaces	A-5
Network Configurations	A-7
Memory Requirements	A-10
Specifications	A-11
Related Products	A-13

Conversion Table

Appendix B

Troubleshooting

Appendix C

What's in This Appendix?	C-1
Interpret the Indicator Lights	C-1
Error Messages from BASIC	C-2
Error Messages from CALL Routines	C-4

**Series A Configuration
Plugs**

Appendix D

Quick Reference

Appendix E

Installing the BASIC Module

What's in This Chapter?

This chapter describes:	On page:
guard against electrostatic discharge	1 -1
unpack the module	1 -2
calculate power requirements	1 -2
install memory module	1 -2
configure the jumpers	1 -3
determine BASIC module placement	1 -10
key the backplane connector	1 -11
install module in the I/O chassis	1 -12
connect peripheral devices	1 -13
power up the module	1 -13
reset the module	1 -13
read the indicator lights	1 -14
what's next?	1 -14

Guard Against Electrostatic Damage

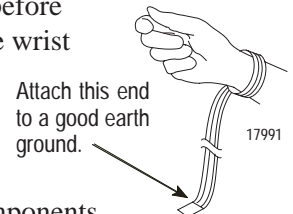
To guard against electrostatic discharge damage, we construct, test, and pack our products in a static-safe environment. We ship products that require individual protection in bags that are sealed and shielded from damage. Equipment that is not packed in a static-shielded bag is adequately protected as long as you keep it assembled.

When you remove a product from a static-shielded bag or disassemble a product, do it in a static-safe environment. We recommend the 3M[®] Type 8005 Portable Field Service Grounding kit or the equivalent for providing a static-safe environment.



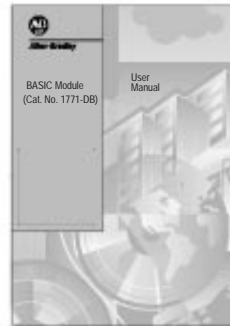
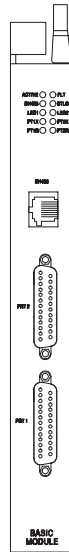
ATTENTION: To avoid damaging the module with electrostatic discharge:

- Wear an approved wrist-strap grounding device or touch a grounded object to discharge yourself before handling any equipment. (Note that the wrist strap is not supplied with the module.)
- Do not touch backplane connectors or connector pins.
- If you configure or replace internal components, do not touch other circuit components inside the module.
- When not in use, keep components in a static-shielded bag.



Unpack the Module

Verify all the items in your package against the packing sheet. If any of the items are missing or incorrect, contact your local Allen-Bradley sales office.



Important: Save packing materials in case you need to return an item for servicing.

Calculate Power Requirements

The BASIC module receives its power through the 1771 I/O chassis backplane from the chassis power supply. The maximum current drawn by the BASIC module is:

- 0.65A – without a device connected to the DH485 port
- 0.75A – with a DH-485 Interface/Converter (1747-PIC) connected to the DH485 port

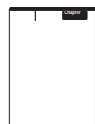
Add this value to the requirements of all the other modules in your I/O chassis to prevent overloading the chassis backplane and/or backplane power supply.



Refer to the documentation accompanying your power supply for additional information.

Install Memory Module

Install your optional memory module into the BASIC module before placing the module into the I/O chassis.

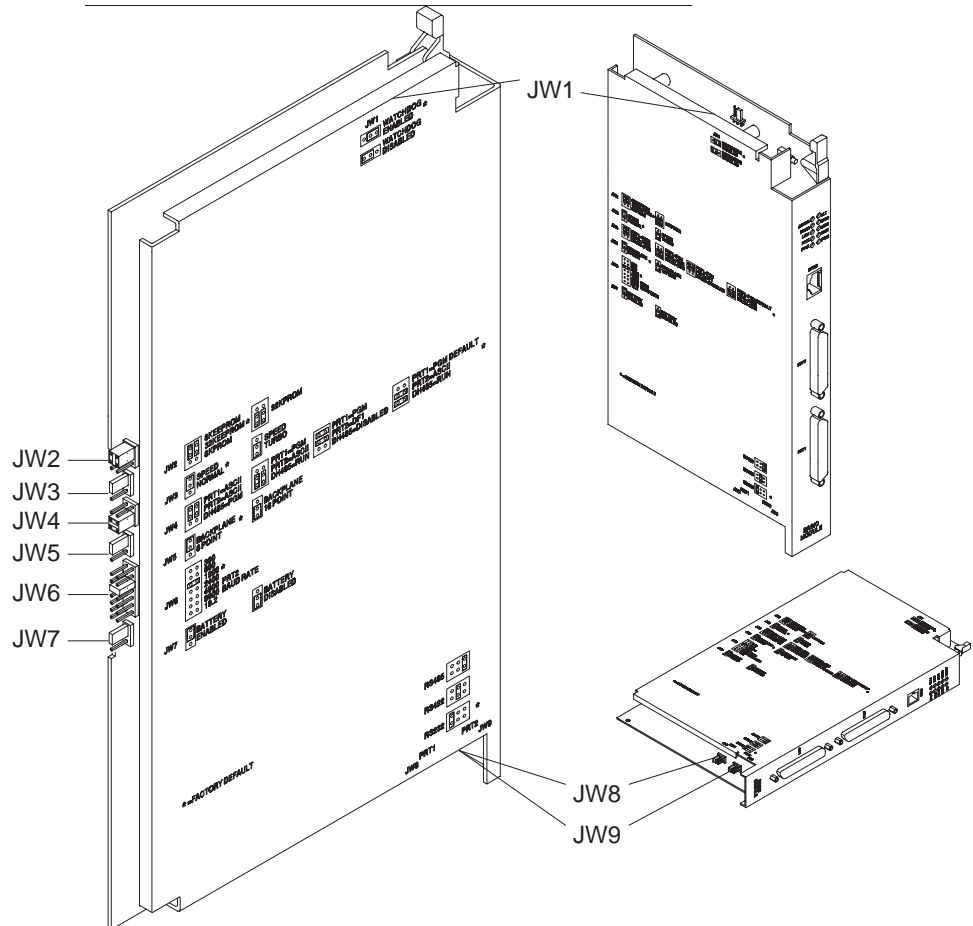


Refer to Chapter 3 “Installing and Replacing Components” for information on how to install the optional memory module.

Configure the Jumpers

The BASIC module has nine sets of jumpers that you need to set. For future reference, place a ✓ next to the jumper setting you choose in the “Your Selection” column of the tables to follow.



Jumper	Description	Page
JW1	watchdog timer configuration	1 -4
JW2	memory module configuration	1 -4
JW3	CPU speed select	1 -5
JW4	operating mode	1 -6
JW5	backplane configuration	1 -7
JW6	PRT2 communication rate select	1 -8
JW7	battery configuration	1 -9
JW8	PRT1 configuration	1 -9
JW9	PRT2 configuration	1 -9



ATTENTION: Do not expose the BASIC module to surfaces or other areas that may typically hold an electrostatic charge. Electrostatic charges can alter or destroy memory and degrade or destroy the sensitive electronic components.



Set Watchdog Timer Enable Jumper (JW1)

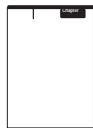
Use JW1 to enable the watchdog timer. Unless you are using assembly language code that you programmed for your Series A module, you should enable this jumper.

To:	Set jumper :	✓ your selection:
enable the watchdog timer		factory setting
disable the watchdog timer		

Set Memory Module Configuration Jumper (JW2)

Use JW2 to configure your non-volatile memory. Jumper JW2 redirects the BASIC module circuitry for the different memory modules. If you are not using a memory module, this jumper is not applicable. The BASIC module runs properly with the jumper in either position.

If you have this PROM :	Set jumper:	✓ your selection:
8K EEPROM		
32K EEPROM		
8K EPROM		
32K EPROM		



Refer to Chapter 3 for more information on memory modules.





ATTENTION: Other jumper settings for JW2 are invalid and may cause damage to the BASIC module.

Set CPU Speed Select Jumper (JW3)



Use JW3 to select the operating speed of your BASIC module processor. Unless you are using a memory module that is slower than 90 ns, set this jumper to turbo to obtain optimum performance. Memory modules that are slower than 90 ns are too slow to run in turbo; you must set the jumper to normal. Refer to Chapter 3 for more information on memory modules.

For this CPU speed:	Set jumper:	✓ your selection:
normal (operating speed of Series A)	 factory setting	
turbo (operating speed approx. 4 times speed of Series A)		

Set Operating Mode Jumper (JW4)



Use JW4 to configure your communication ports as a program port, ASCII port, network port, or DF1 protocol. Also configure your power on operating condition. Refer to Chapter 2 for more information regarding the operation mode.

If PRT1 is:	and PRT2 is:	and DH485 is:	and action at power up is:	Set jumper:	✓ your selection:
ASCII port ^②	ASCII port ^②	program port ^{②④}	executing program ^①		
program port ^②	ASCII port ^②	network port ^②	executing program ^①		
			in Command mode	factory setting 	
	DF1 protocol ^②	disabled	executing program ^①		

^① If you previously executed a PROG2 (page 10 -13).

^② Mode settings of these ports determined by information you saved with a PROG1 (page 10 -12), PROG2, or MODE (page 11 -20).

^③ Operating with default communication settings (1200, n, 8, 1, s). You may want to use this mode for troubleshooting purposes.



^④ You must use PBASE software (1747-PBASE) if using the DH-485 port as the program port.

Set Backplane Configuration (JW5)

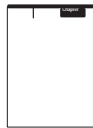
Use JW5 to set the BASIC module backplane configuration. The BASIC module can perform both block and discrete transfers. With JW5 set to 8-point mode, the BASIC module uses 8 bits in both the input and output image table for block transfer. When you have JW5 (page 1 -7) configured for 16-point mode, the firmware also allows you to examine/use bits 10–17 for status of the communication ports.

Important: When the BASIC module is in 16-point mode you must perform all block-transfers synchronously with a PLC-5 processor. Asynchronous block transfers do not work with this configuration and cause the backplane circuit within the module to lock up. The BASIC module ships with JW5 configured for 8-point mode (Series A compatible). Unless you are using calls 32, 33, 34, 49, 50, 118, 122, or 123 (see Chapters 12 and 13) you should be in 8-point mode.

Important: You cannot use 2-slot chassis addressing if this jumper is set for 16-point mode. See page 1 -10 for information regarding the backplane configuration and addressing.

Input image bits	Output image bits	Read block words	Write block words	Jumper setting	✓ your selection:
8	8	64 max	64 max	 factory setting (Series A setting)	
16	16	64 max	64 max	 ①	

① 2-slot chassis addressing not allowed. Block transfers must be synchronous.



Refer to Chapter 5 for more information regarding the backplane mode.

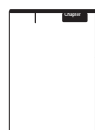
Set PRT2 Communication Rate Select Jumper (JW6)

Use JW6 to set the communication rate for PRT2 at power-up. Set the communication rate according to your application.

Important: You can also select the communication rate for PRT2 within your program. The settings you select with PROG1 (page 10 -12) and PROG2 (page 10 -13) override the jumper setting until the module is powered down. The settings you select with MODE (page 11 -20) override both the jumper and the PROG1 and PROG2 settings until the module is powered down. If you use PROG1, PROG2, or MODE with the “E” option storage to save the port settings in the user EEPROM, the unit then powers up with the stored configuration.

To set the communication rate for:	Set jumper:	✓ your selection:
300 bit/s		
600 bit/s		
1200 bit/s		
2400 bit/s		
4800 bit/s		
9600 bit/s		
19.2k bit/s		



factory setting



Refer to Chapter 2 for more information regarding communication rates.




Set Battery Enable Jumper (JW7)

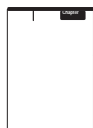
Use JW7 to enable the battery. To conserve the battery, your module is shipped with the battery disabled. When the BASIC module is in use, you should enable the battery. If you do not enable the battery, your program is not backed up if a power failure occurs.

To:	Set jumper:	✓ your selection:
enable the battery		
disable the battery		factory setting

Set PRT1 and PRT2 Configuration Jumpers (JW8 and JW9)

Use JW8 to configure the communication mode for the PRT1 port. Use JW9 to configure the communication mode for the PRT2 port. Set your jumpers to the correct communication based on the device you want to connect to the BASIC module.

For this communication:	Set jumper:	✓ your selection:
RS485		
RS422		
RS232		factory setting



Refer to Chapter 2 for more information regarding communication settings.

Determine BASIC Module Placement

You install the BASIC module in a 1771-I/O chassis. You can place your module in any slot of the I/O chassis except for the extreme left slot (his slot is reserved for processors or adapter modules). We recommend that you remember these points:

- When selecting slots for modules, always try to group modules to minimize adverse effects from electrical noise and radiated heat.
- Group analog and low-voltage dc modules away from ac modules or high-voltage dc modules to minimize electrical noise interference.



Important: Certain processors restrict the placement of block-transfer output modules. Refer to the user manual for your particular processor for more information.

Refer to this table to determine where the BASIC module fits into your module group and address scheme. For more information on jumper JW5, see Set Backplane Configuration on page 1 -7.

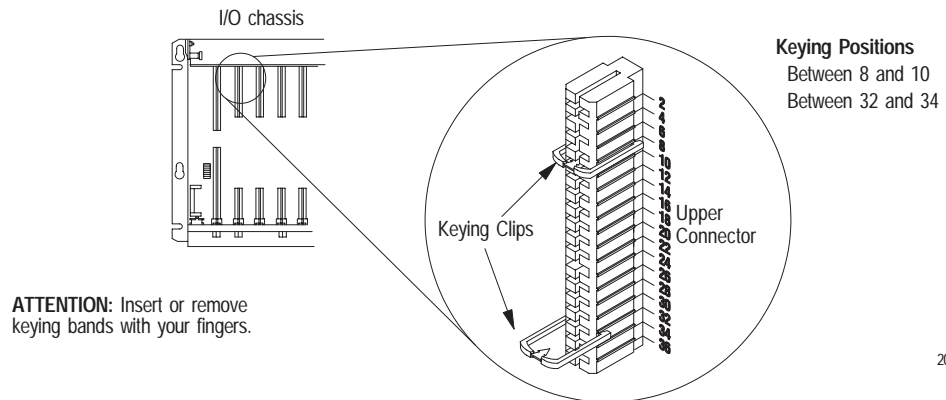
If using this chassis addressing:	If JW5 is set to 8-pt. mode:	If JW5 is set to 16-pt. mode:
2-slot	place the BASIC module in any module group with any 8-bit discrete or block transfer module	not permitted
1-slot	place the BASIC module in any module group with any 8-bit, 16-bit, discrete or block transfer module	place the BASIC module in any module group with any 8-bit, 16-bit discrete or block transfer module
1/2-slot	no restrictions	no restrictions

Key the Backplane Connector

Use the plastic keying clips shipped with each I/O chassis to key the I/O slot to accept only a BASIC module.

The BASIC module is slotted in two places on the rear edge of the circuit board. The position of the keys on the backplane connector must correspond to these slots to allow insertion of the module.

Snap the keys onto the upper backplane connectors between 8 and 10 and between 32 and 34. (Same as Series A)



You can change the position of these keys if subsequent system design and rewiring makes insertion of a different type of module necessary.



ATTENTION: Observe these precautions when inserting or removing keying clips:

- Insert or remove keying clips with your fingers.
- Make sure that keying placement is correct.

Incorrect keying or the use of a tool can result in damage to the backplane connector and possible system faults.

Install the Module into the 1771 I/O Chassis

You are now ready to install the module into the I/O chassis.



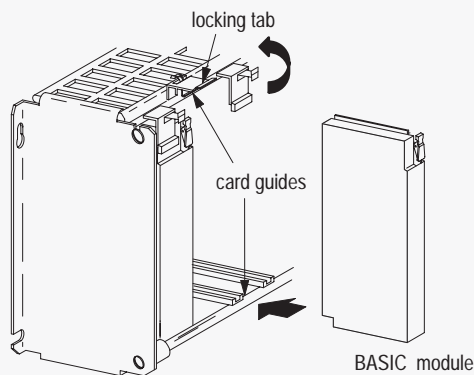
ATTENTION: Disconnect and lockout all power from the programmable controller and system power supplies before installing modules to avoid injury to personnel and damage to equipment.

1. Turn off power to the I/O chassis.
2. Use the card guides on the top and bottom of the slot to place the BASIC module into position.

Important: Apply firm even pressure on the module to seat it into its backplane connector.

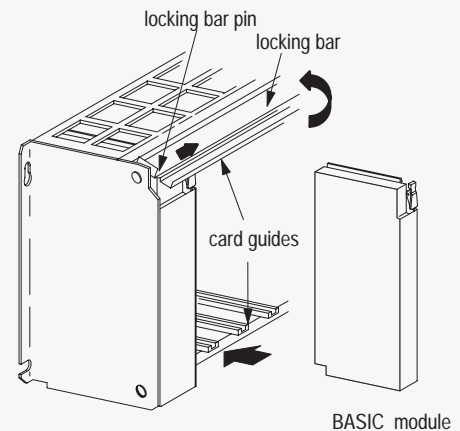
3. Secure the module depending on the of I/O chassis you have:

- 3a.** Snap the chassis latch over the top of the module to secure it.



1771-A1B, -A2B, -A3B, -A3B1, -A4B I/O chassis

- 3b.** Swing the chassis locking bar down into place to secure the modules. Make sure the locking pins engage.



1771-A1B, -A2B, -A3B1, -A4B Series B I/O chassis

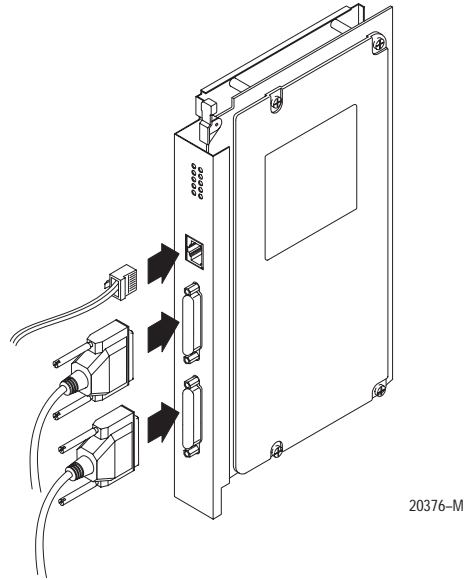
19809

Connect Peripheral Devices

Now that you have installed your BASIC module into the I/O rack, you need to connect your external devices to the communication ports.



See Chapter 2 for cable pin out information.

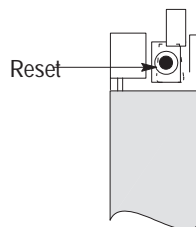


Power up the Module

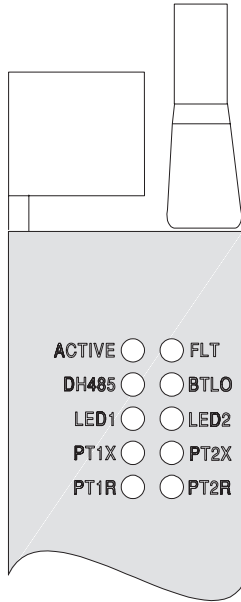
Apply power to your I/O chassis backplane. At power up, the top six LEDs are on. They go off one at a time as each part of the module self-test successfully passes. If a fault occurs at power up, the LEDs give an indication of the portion of the test that failed. Depending on how you set jumper JW4 (page 1 -6) and if you've executed a PROG2 (page 10 -13), the BASIC module powers up either executing a program or in Command mode.

Reset the Module

Although not required during initial power up of the module, the BASIC module has a hard reset switch located behind the module ejector tab. When this switch is pressed (you may need to use a pointed instrument to press it), the BASIC module initiates a full reset. The BASIC module reacts to this reset the same as it does when you turn on power to your I/O chassis backplane.



Read the Indicator Lights



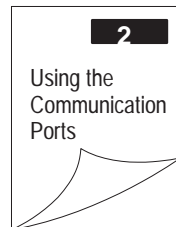
The BASIC module has 10 indicator LEDs:

This LED:	Indicates:
ACTIVE	the module mode and whether the BASIC module is receiving power from the backplane
FLT	whether a system problem was detected during background diagnostics
DH485	whether port DH485 on the BASIC module is active for communication
BTLO	whether the voltage of the battery that backs up RAM is low
LED1	User definable. LED activated through the user program.
LED2	User definable. LED activated through the user program.
PT1X	whether port PRT1 on the BASIC module is transmitting signals.
PT2X	whether port PRT2 on the BASIC module is transmitting signals
PT1R	whether port PRT1 on the BASIC module is receiving signals
PT2R	whether port PRT2 on the BASIC module is receiving signals



Refer to Appendix C for more information on troubleshooting.

What's Next?



- Overview
- Communication Modes
- Hardware Handshaking
- Communication Rates
- Operating Modes

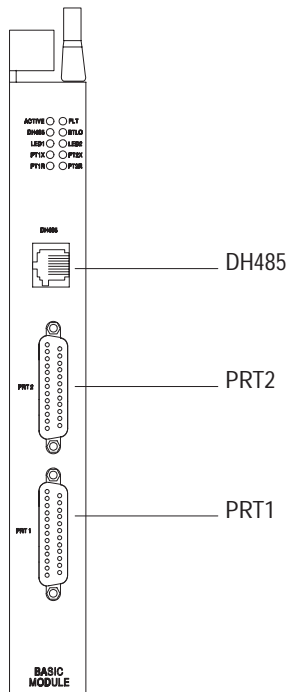
Using the Communication Ports

What's in This Chapter?

This chapter describes:	On page:
communication ports overview	2 -1
communication modes	2 -2
handshaking	2 -4
communication rates	2 -6
operating modes	2 -7
what's next?	2 -13

Communication Ports Overview

The BASIC module has three communication ports: DH485, PRT2, and PRT1. Through the configuration you select, you can designate at least one of these ports to function as a program port, ASCII port, network port, or DF1 protocol port.



Communication Modes

PRT1 and PRT2 Port

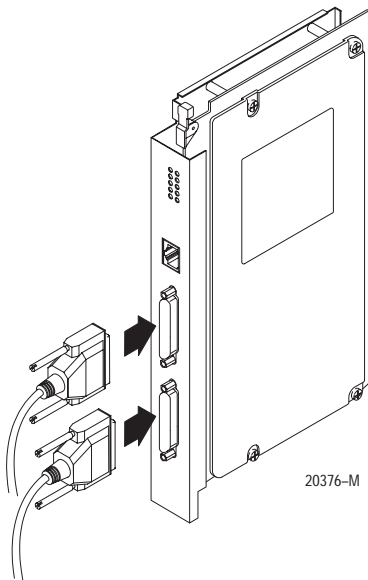
You can configure ports PRT1 and PRT2 for these communication modes:

- RS-232C – communicate with a RS-232 device or an unterminated RS-423 device within 50 ft.
- RS-422 – point to point and multidrop for RXD/TXD connections
- RS-485 – multidrop supported for RXD/TXD connections

The communication mode you choose depends on the device you are connecting to the BASIC module. Refer to the documentation accompanying the device. JW8 sets the communication mode for port PRT1 (page 1 -9) and JW9 sets (page 1 -9) the communication mode for port PRT2 .

PRT1 and PRT2 Pinout Connections

PRT1 and PRT2 have a DB25 female connector. Here is the pinout for the connectors:



Pin	RS-232	RS-422	RS-485
1	chassis/shield	chassis/shield	chassis/shield
2	TXD	N/A ^②	N/A ^②
3	RXD	N/A ^②	N/A ^②
4	RTS	N/A ^②	N/A ^②
5	CTS	N/A ^②	N/A ^②
6	DSR	N/A ^②	N/A ^②
7	common	common	common
8	DCD	N/A ^②	N/A ^②
9	common	common	common
10	common	common	common
14	N/A ^①	TXD	TXD/RXD
16	N/A ^①	RXD	N/A ^②
18	N/A ^①	RXD'	N/A ^②
20	DTR	N/A ^②	N/A ^②
25	N/A ^①	TXD'	TXD'/RXD'

^① These pins are not a No Connection (N/C). In RS-232 mode, the RS-422 and RS-485 load is still present and should not be connected to any device in this mode.

^② In RS-422 and RS-485 modes, these pins are still connected to their RS-232 drivers and receivers. Do not use these pins in either RS-422 or RS-485 mode.

Important: Pins 11, 12, 13, 15, 17, 19, 21, 22, 23 and 24 are a No Connection (N/C)

PRT1 and PRT2 are electrically isolated to 500V dc.

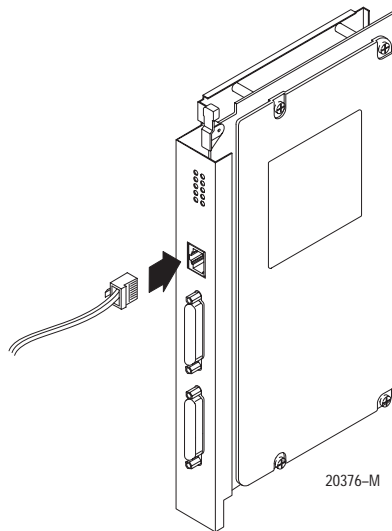
PRT1 and PRT2 Transmit and Receive Buffers

Ports PRT1 and PRT2 each have a 256-character receive (input) buffer and a 256-character transmit (output) buffer. Data in these buffers are monitored by circular queues. If a queue detects that a buffer is full (i.e., the buffer has 256 characters waiting to be serviced), and handshaking is enabled (software or hardware), then the queue does not allow new characters to enter the buffer until existing characters have been serviced and room is available for the new characters.

DH485 Port

The DH485 port uses the DH-485 communication mode to communicate with SLC processors. The DH485 port has an RJ-45 connector. Use the standard Allen-Bradley cables (cat. no. 1747-C10, 1747-C11, 1747-C20) for all DH-485 interconnects.

The DH485 port is not electrically isolated.



Handshaking

The BASIC module support both hardware and software handshaking. You turn hardware and software handshaking on and off through the MODE statement (page 11 -20).

Software Handshaking

The BASIC module uses these rules when software handshaking is enabled:

- When the BASIC module receives an XOFF from the external device, it is recognized immediately and the BASIC module stops sending characters from the transmit buffer to the UART. However, if there are any characters already in the UART, these characters are sent to the external device. The BASIC module continues sending new characters to the transmit buffer (if PRINT statements are executed). If the transmit buffer becomes full (i.e., 256 characters waiting to be transmitted before it receives an XON), then BASIC pauses (while executing the PRINT statement) until space is available in the transmit buffer.
- The BASIC module constantly monitors the number of characters in its receive buffers. If the receive buffer accumulates 192 characters before any are serviced by the BASIC program, it sends an XOFF to the external device. The safety margin of 64 characters in the buffer ensures the BASIC module can accommodate additional characters in case the external device does not acknowledge the XOFF immediately and continues to transmit. When characters are serviced by the BASIC program and the receive buffer contains less than 128 characters, the BASIC module sends an XON to the external device to allow new characters to be received.

Important: The BASIC module automatically initiates XON/OFF codes when software handshaking is enabled. Do not attempt to generate these codes from the BASIC program or you could cause the serial ports to appear to be “locked-up.”

Important: If you are receiving non-ASCII data from an external device, we recommend that you do not use software handshaking. If software handshaking is enabled and the external device happens to transmit data that is equivalent to the code for XOFF, the BASIC module stops transmitting the characters and appears to be “locked-up.”

Hardware Handshaking

The BASIC module uses these rules when hardware handshaking is enabled. The BASIC module:

- does not transmit until CTS, DCD, and DSR become active
- examines DSR and DCD following the receipt of a character. If the DSR and DCD are active, the character is placed in the input queue. If DSR or DCD is inactive, the character is assumed to be noise and is discarded.

Important: You need to know whether the device connecting to the BASIC module has a DTE or DCE interface.

The BASIC module serial ports are configured as 25-pin Data Terminal Equipment (DTE), as are most terminals or computer ports.

DTE 25 pinout	Signal description	Signal from DTE perspective	DTE 9 pinout	Signal description
2	TXD–Transmitted Data	Output	3	
3	RXD–Received Data	Input	2	
4	RTS–Request to Send	Output	7	
5	CTS–Clear to Send	Input	8	
6	DSR–Data Set Ready	Input	6	
7	Com–Signal Common	Shared	5	
8	DCD–Data Carrier Detect	Input	1	CD–Carrier Detect
20	DTR–Data Terminal Ready	Output	4	
22	NC–No Connection (for BASIC module only)	Input	9	RI–Ring Indicator (BASIC module does not support)

Devices such as modems are Data Communication Equipment (DCE). Pinouts on these terminals are defined for ease of interfacing with DTE equipment.

DCE 25 pinout	Signal description	Signal from DC perspective	DCE 9 pinout
2	TXD–Transmitted Data	Input	3
3	RXD–Received Data	Output	2
4	RTS–Request to Send	Input	7
5	CTS–Clear to Send	Output	8
6	DSR–Data Set Ready	Output	6
7	Com–Signal Common	Shared	5
8	CD–Carrier Detect	Output	1
20	DTR–Data Terminal Ready	Input	4
22	RI–Ring Indicator	Output	9

Important: All signal directions that are listed in the previous two tables are valid. For example, TXD, Transmitted Data, is a DTE output but is also a DCE input. The signal description is the same for both the DTE and DCE but the direction of the signal (perspective) has changed based on whether you have a DTE or DCE device.

Communication Rates

You can operate PRT1 and PRT2 ports full-duplex and DH485 port half-duplex at 300, 600, 1200, 2400, 4800, 9600 or 19200 bit/s. You can set the communication rates for PRT1, PRT2, and DH485 ports using the MODE (page 11 -20) statement and store the settings using the PROG1 (page 10 -12) and PROG2 (page 10 -13) commands. You can set the communication rate for the program port with CALL 78 (page 13 -8).

Important: Jumper JW6 sets the communication rate on the hardware for PRT2 (page 1 -8). You can also select the communication rate for PRT2 within your program. The settings you select before using PROG1 or PROG2 override the jumper setting until the module is powered down. The settings you select with MODE overrides both the jumper and the PROG1 and PROG2 settings until the module is powered down. If you use the PROG1, PROG2 or MODE to store the port settings in the user EEPROM, the unit then powers up with stored configuration.

Important: The BASIC module cannot transmit and receive continuous streams of data at 19.2k bit/s on all three ports at the same time.

Here is the communication rates vs distance:

Communication Rate (bit/s)	Maximum distance allowed in meters (feet) for:			
	RS-232	RS-423	RS-422	RS-485
300	15 (50)	15 (50)	1230 (4000)	1230 (4000)
600	15 (50)	15 (50)	1230 (4000)	1230 (4000)
1200	15 (50)	15 (50)	1230 (4000)	1230 (4000)
4800	15 (50)	15 (50)	1230 (4000)	1230 (4000)
9600	15 (50)	15 (50)	1230 (4000)	1230 (4000)
19200	15 (50)	15 (50)	1230 (4000)	1230 (4000)

Important: Use the RS-232 jumper settings for JW8 or JW9 when communicating in RS-423 mode (page 1 -9). RS-423 devices should be unterminated and cable length should be a maximum of 50 ft.

Operating Modes

Depending on how you set jumper JW4 (see Set Operating Mode, page 1 -6), you can configure:

Port PRT1 as:

- ASCII port
- program port

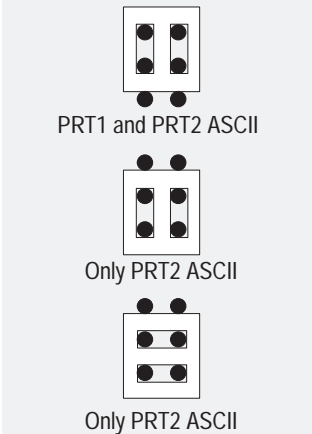
Port PRT2 as:

- ASCII port
- DF1 protocol port

Port DH485 as:

- program port
- network port
- disabled

JW4 ASCII Port Configurations



ASCII Port

If you set JW4 to one of the configurations shown at the left, PRT1 and/or PRT2 are ASCII ports (asynchronous serial communication channels) compatible with RS-232C, RS-422, RS-485 interfaces. When you configure PRT1 and PRT2 as ASCII ports, you use jumpers JW8 and JW9 (page 1 -9) to select an electrical interface. The RS-485 electrical interface is not the same as the DH-485 network (the electrical interface is similar but the DH-485 network has embedded firmware to control communication). Through the ASCII ports you can interface with:

- printers
- terminals
- commercial asynchronous modems

To interface with these devices, use either bi-directional XON and XOFF software handshaking or RTS/CTS, DTR, DSR, DCD hardware handshaking.

Use the **MODE** command (page 11 -20) to change ASCII port configuration. You can also use **CALL 30** (page 12 -20) to change the ASCII port configuration for PRT2 only.

The ASCII port has these parameters:

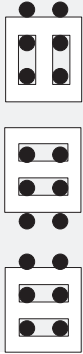
Port parameters	Selections	Default settings
communication rate	300, 600, 1200, 2400, 4800, 9600, 19200	1200
parity	none (N), even (E), odd (O)	N
number of data bits	7 or 8	8
number of stop bits	1 or 2	1
handshaking	no handshaking (N) software handshaking (S) hardware handshaking (H) hardware and software handshaking (B)	S
storage type	store information in user ROM and RAM (E) store information in battery backed RAM (R)	R

When you select 8 bits/character you have full access to all 8 bits of each character on both input and output data bytes.

Program Port

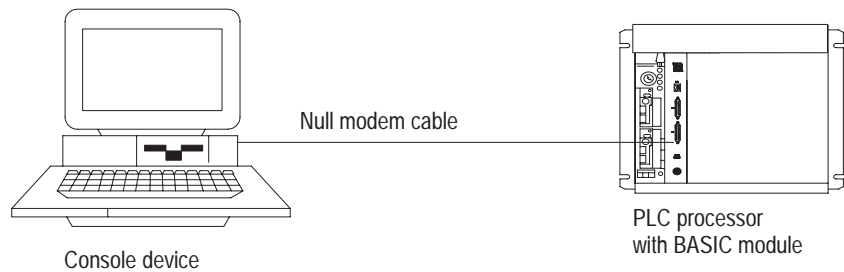
You can configure either PRT1 or DH485 as your program port.

JW4 PRT1 Program Port Configurations
(See chapter 1 for additional information on these settings.)



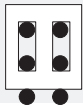
PRT1 Configured as Program Port

If you set JW4 to one of the configurations shown at the left, PRT1 is the program port. In this configuration, the serial port on the console device is connected to port PRT1 on the BASIC module. The console device communicates with the BASIC module through terminal emulation over an RS-232 interface.



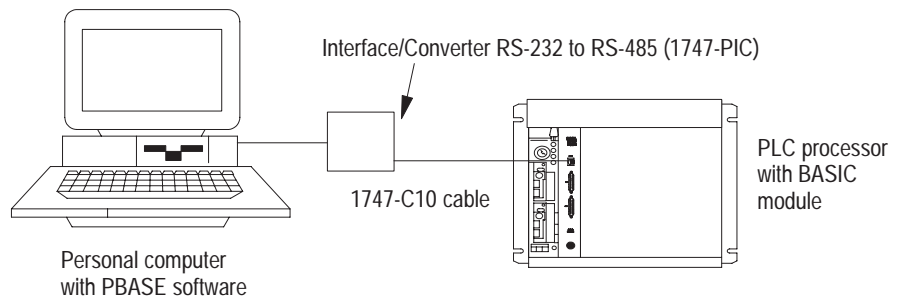
Important: When using PBASE to interface with the RS-232 port of the BASIC module, PBASE must be configured for RS-232 communication through the configuration and terminal selection menus. Refer to the BASIC Development Software Programming Manual (publication number 1746-6.2).

JW4 DH485 Program Port Configuration



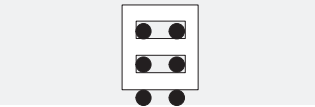
DH485 Configured as Program Port

If you set JW4 to the configuration shown at the left, port DH485 is the program port. In this configuration, the serial port on the personal computer interfaces with port DH485 on the BASIC module through a 1747-PIC Interface/Converter. The 1747-PIC Interface/Converter converts the RS-232 signals from the personal computer RS-232 serial port to RS-485 format. When DH485 is the program port you must use PBASE to program the BASIC module.



Important: You must use PBASE to use the DH485 port as a program port. Configure PBASE for DH-485 communication through the configuration and terminal selection menus. Refer to the BASIC Development Software Programming Manual (publication number 1746-6.2) for additional information.

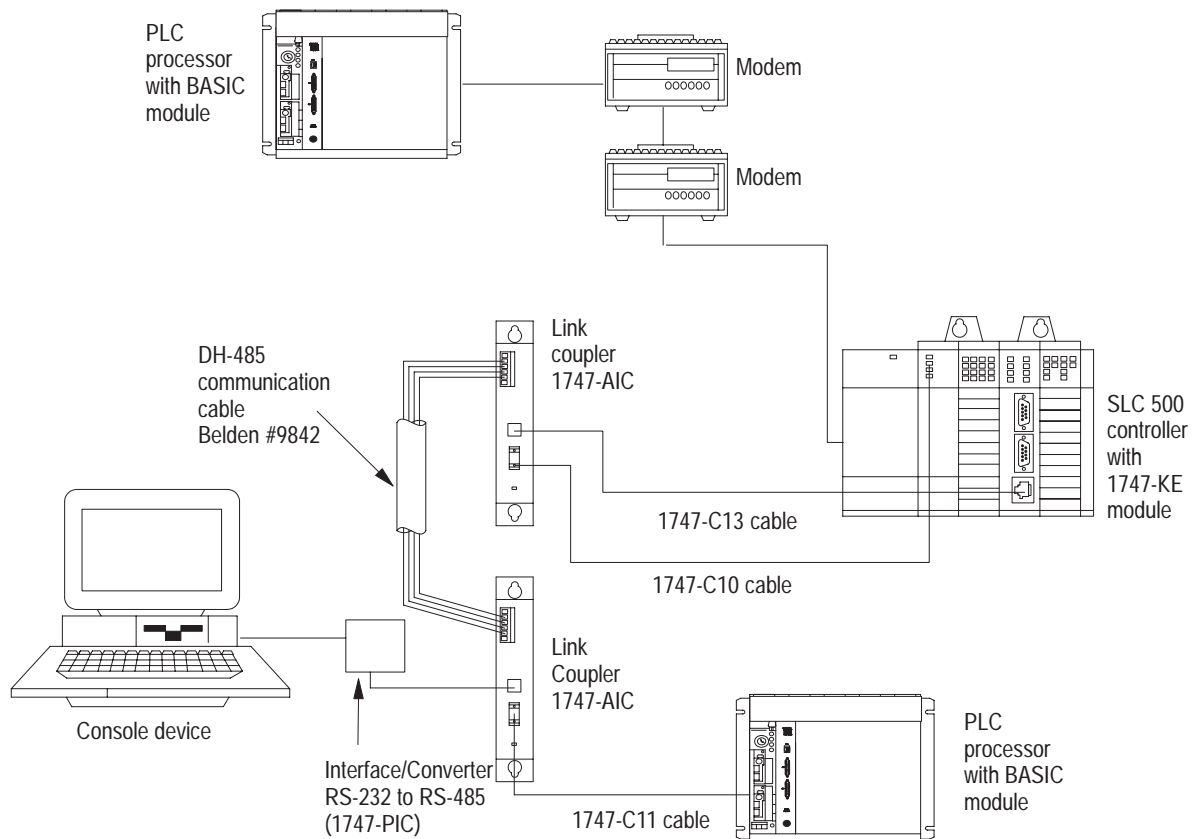
JW4 DF1 Protocol Configuration



DF1 Protocol

If you set JW4 to the configuration shown at the left, PRT2 port can be configured via a BASIC program for DF1 protocol. The BASIC module uses DF1 protocol to communicate with external devices using, for example, a leased phone line, radio link or dial-up modem.

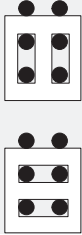
Important: When DF1 protocol is selected on port PRT2, DH-485 communications are disabled.



Important: The 1747-C13 cable acts only as a communication link and does not carry 24V dc power. The 1747-C10, 1747-C11, 1747-C20 cable carries 24V dc power from the processor to the link coupler. (The 1747-C10 cable, 1747-C11 and 1747-C20 cables are interchangeable.)

Important: If the modems are dial-up, the BASIC program may initiate dial-up and then switch port PRT2 to DF1 protocol when connection is made to the 1747-KE or 1770-KF3 DH-485 Communication Interface Module. JW4 must still be set for DF1 protocol; however, the port is not active until you enable it with CALL 108 (page 13 -38).

JW4 DH485 Network Port Configuration
(See chapter 1 for additional
information on these settings.)



Network Port

If you set JW4 to one of the configurations shown at the left, your BASIC module can be interfaced with a DH-485 network using a combination of:

- 1747-AIC isolated link coupler
- 1747-PIC interface/converter
- 1784-KR DH-485 interface card
- 1770-KF3 DH-485 communication interface module
- 1747-KE DH-485/RS-232C communication interface module

The BASIC module and SLC processor CPU act as two separate nodes on the DH-485 network.

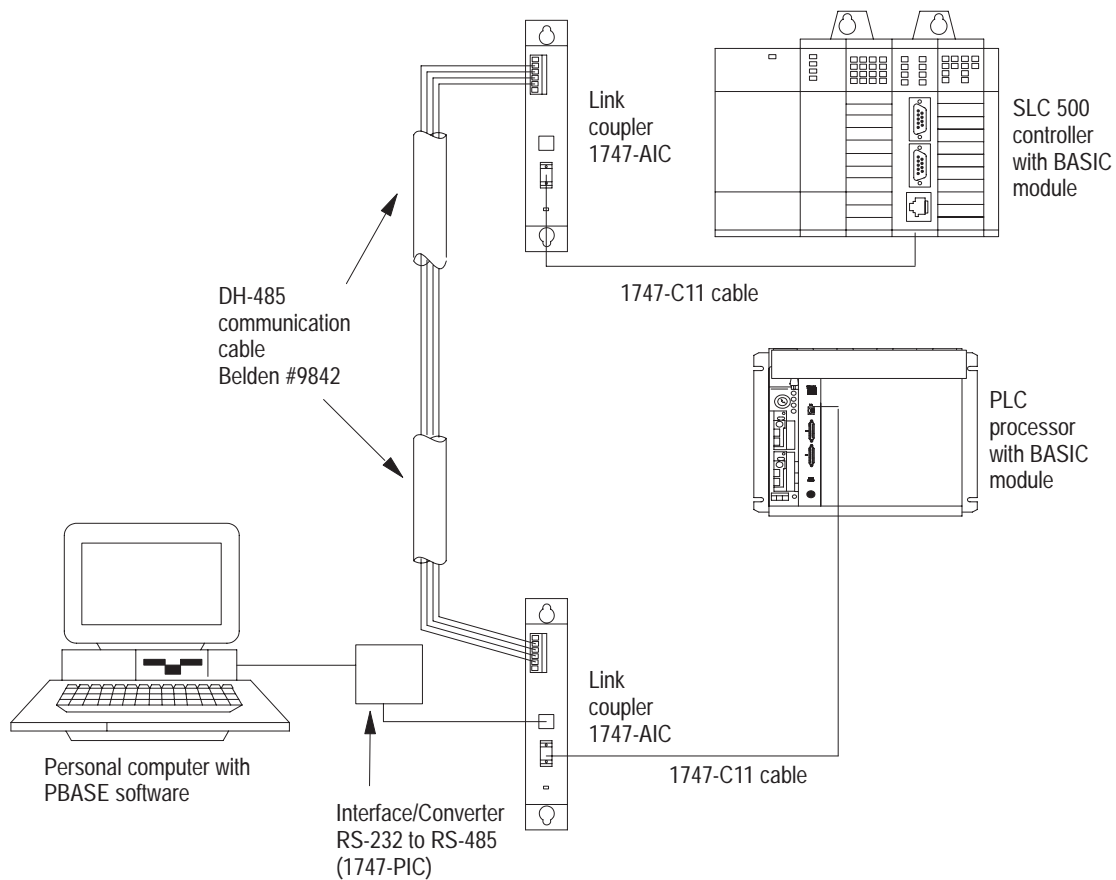
Cable Requirements

Use the 1747-C10 cable, 1747-C11 cable, 1747-C13 cable, or 1747-C20 cable to interface port DH485 of the BASIC module with a 1747-AIC link coupler. Use the DH-485 communication cable, Belden #9842, to interface between the link couplers on the DH-485 network. The 1747-C13 cable acts only as a communication link and does not carry 24V dc power. The 24V dc can come from either the processor or an outside power source. The 1747-C10, 1747-C11, or 1747-C20 cable carries 24V dc power from the processor to the link coupler. (The 1747-C10 cable, 1747-C11 cable and 1747-C20 cable are interchangeable.) A cable connected to the outside power source carries 24V dc from the outside power source to the link coupler.

1747-PIC Interface/Converter/1747-AIC Isolated Link Coupler

Use the 1747-PIC interface/converter to convert the RS-232 signals from the personal computer's serial port to DH-485 format. This figure shows the interface/converter integrating a personal computer with the PBASE software to the BASIC module across a DH-485 network.

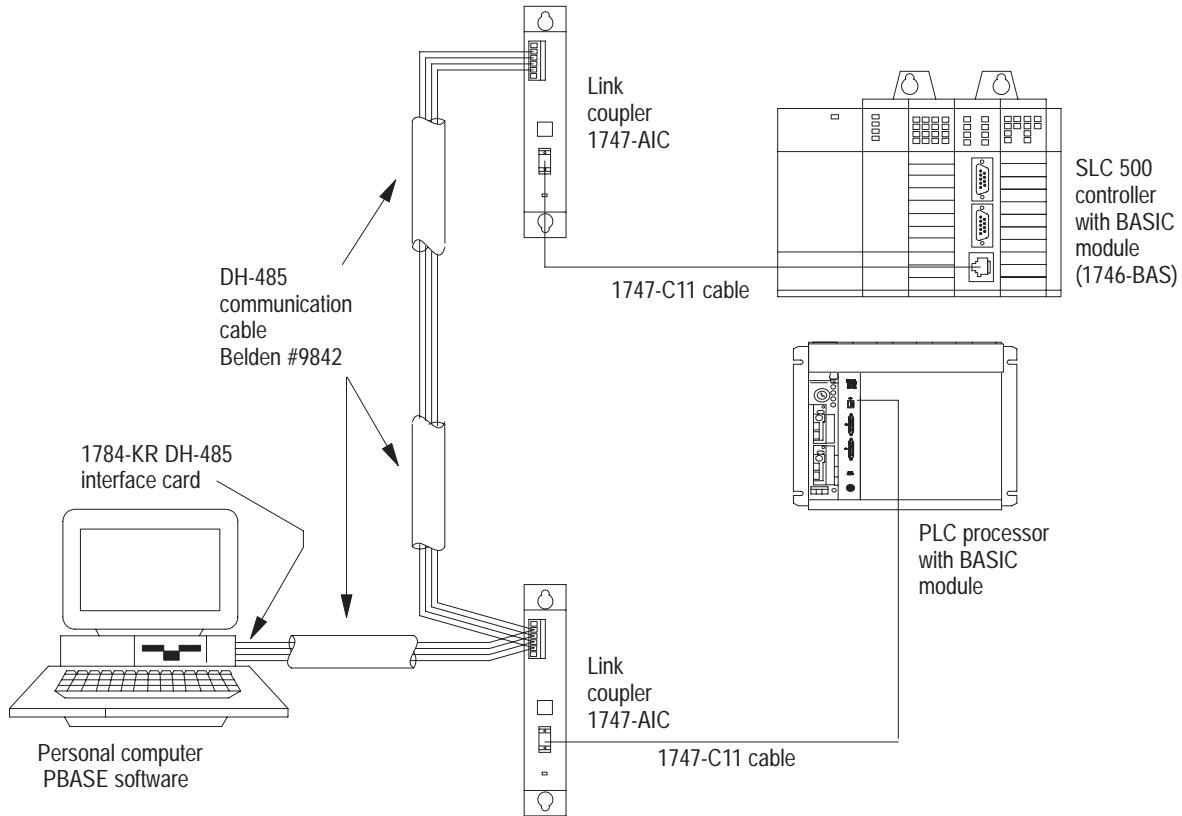
The 1747-AIC isolated link coupler allows you to link modules to the DH-485 network.



Important: When using PBASE to interface with the BASIC module through the 1747-PIC, the BASIC development software must be configured for DH-485 communication through the configuration and terminal selection menus. Refer to the BASIC Development Software Programming Manual (publication number 1746-6.2) for additional information.

1747-AIC Link Coupler/1784-KR DH-485 Interface Card

The 1784-KR DH-485 Interface Card enables your personal computer to communicate across the DH-485 network to the BASIC module without the interface/converter. This figure shows a DH-485 network configuration with the 1784-KR DH-485 Interface Card and its host computer linked with the BASIC module through a link coupler.



In this configuration, your personal computer must have the 1784-KR DH-485 Interface Card installed in one of its expansion slots. The DH-485 data link connector on the 1784-KR card and port DH485 on your BASIC module are interfaced with the DH-485 network through a 1747-AIC link coupler.

What's Next?



- Remove Module from Chassis
- Disassemble Module
- Install Memory Module
- Replace Battery
- Reassemble Module

Notes:

Installing and Replacing Components

What's in This Chapter?

Refer to this chapter if you are installing or replacing a memory module or the battery. If not, go on to Chapter 4, “Programming the BASIC Module”.

This chapter describes:	On page:
before you begin	3 -1
remove the BASIC module from the I/O chassis	3 -2
disassemble the BASIC module	3 -3
install memory module	3 -4
install the battery	3 -8
reassemble the BASIC module	3 -11
what's next?	3 -11

Before You Begin

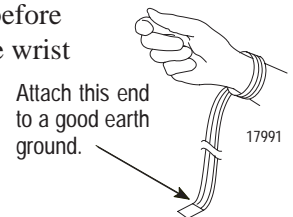
Before you can install the memory module or battery you must remove the module from the I/O chassis and disassemble the module. You need these tools to do this:

- Phillip’s head screwdriver
- flat head screwdriver (for battery)
- chip insertion/extraction tool (for memory modules without a carrier)



ATTENTION: To avoid damaging the module with electrostatic discharge:

- Wear an approved wrist-strap grounding device or touch a grounded object to discharge yourself before handling any equipment. (Note that the wrist strap is not supplied with the module.)
- Do not touch backplane connectors or connector pins.
- If you configure or replace internal components, do not touch other circuit components inside the module.
- When not in use, keep components in a static-shielded bag.



Remove the BASIC Module from the I/O Chassis

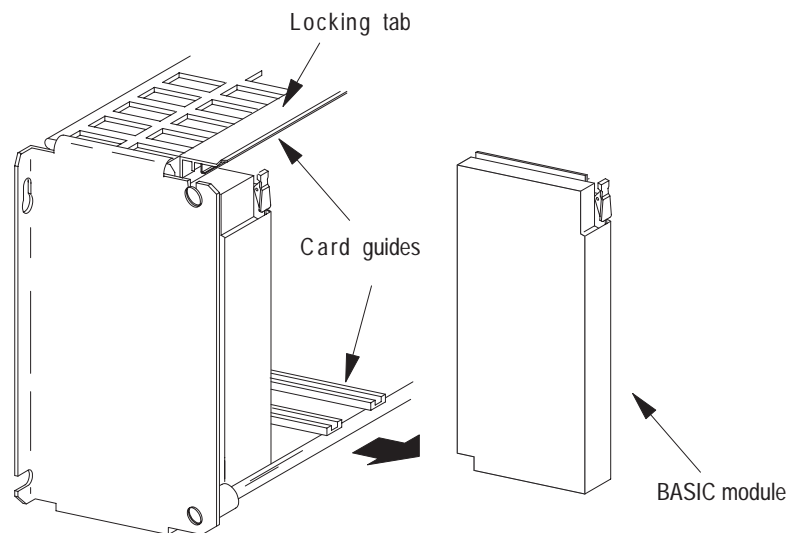
Before you can add or replace components, you must remove the module from the I/O chassis. Go to page 3 -3, if you already removed the BASIC module from the chassis.



ATTENTION: Shut off power to the I/O chassis before removing the BASIC module; otherwise, personal injury or damage to equipment may result.

To remove the BASIC module from the I/O chassis:

1. Remove power from the I/O chassis containing the BASIC module.
2. Put on the wrist strap and ground it to the I/O chassis.
3. Disconnect all cables from the BASIC module.
4. Lift locking tabs and slide the BASIC module out of the I/O chassis.



19466

Disassemble the BASIC Module

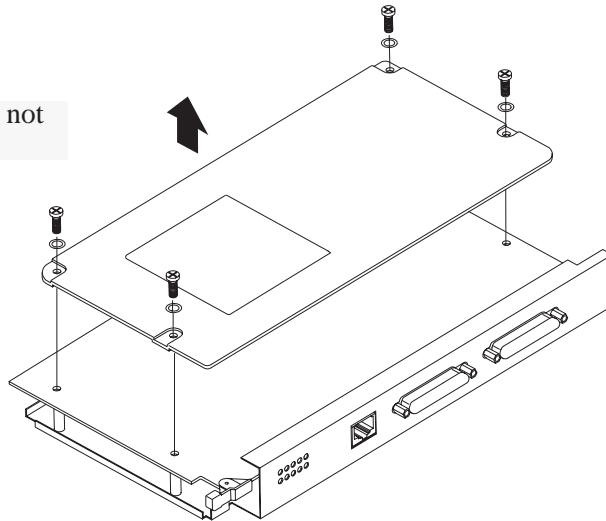
Before you can install the optional memory module or battery, you have to disassemble the BASIC module.



ATTENTION: Electrostatic discharge can damage integrated circuits or semiconductors in the module if you touch backplane connector pins. Use a static-safe workstation, if available.

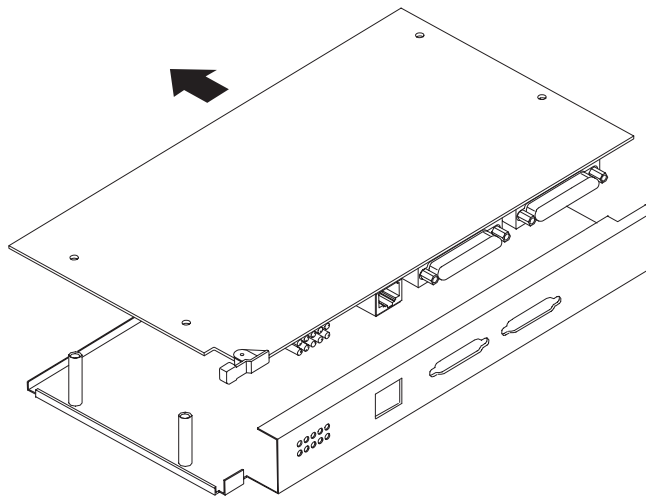
1. Remove the cover from the right side of the BASIC module.

Important: Be careful not to lose the washers.



20369-M

2. Remove the board from the main cover.



20370-M

Install Optional Memory Module

The BASIC module supports these Allen-Bradley Memory Modules:

- 8K EEPROM (Cat. Nos. 1771-DBMEM1 or 1747-M1)
- 32K EEPROM (Cat. Nos. 1771-DBMEM2 or 1747-M2)
- 8K EPROM (Cat. No. 1747-M3)
- 32K EPROM (Cat. No. 1747-M4, PN 940654-02, or PN 940654-03)

Also, you can use any JEDEC standard 8K, 16K, or 32K EPROM or 8K or 32K EEPROM with speeds faster than 150 ns (for example 90 ns). When using turbo mode, the memory modules must have speeds of 90 ns or faster (for example 50 ns).

You can store up to 255 programs in memory modules. The programs are stored in sequence in the PROM for retrieval and execution. The BASIC module generates all of the timing signals needed to program most EEPROM optional memory modules. However, EPROM optional memory modules must be programmed by an external PROM programmer. Jumper JW2 is used to redirect the module circuitry for the different memory module options (see Chapter 1). These commands allow you to generate and manipulate the PROM file: RAM, ROM, XFER, PROG, PROG1, PROG2 (Chapter 10).



The data format of the BASIC module EEPROM and EPROM optional memory modules is hexadecimal. PBASE software provides a hex file transfer option that can be used to upload and download hex files to the BASIC module EEPROM or EPROM. The primary use of hex file transfers is to transfer the data from an EEPROM in one BASIC module to an EEPROM in another BASIC module. Hex file transfers can also be used to copy the data of an EEPROM to a EPROM via a PROM programmer.

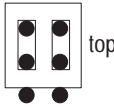

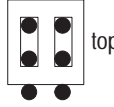

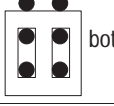

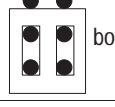



Refer to the BASIC Development Software Programming Manual (publication number 1746-6.2) for additional information on hex file transfers.

Important: The BASIC module can program and erase EEPROM memory modules. However, it cannot program or erase EPROM memory modules.

To install your optional memory module:

Refer to this table when installing your memory module.

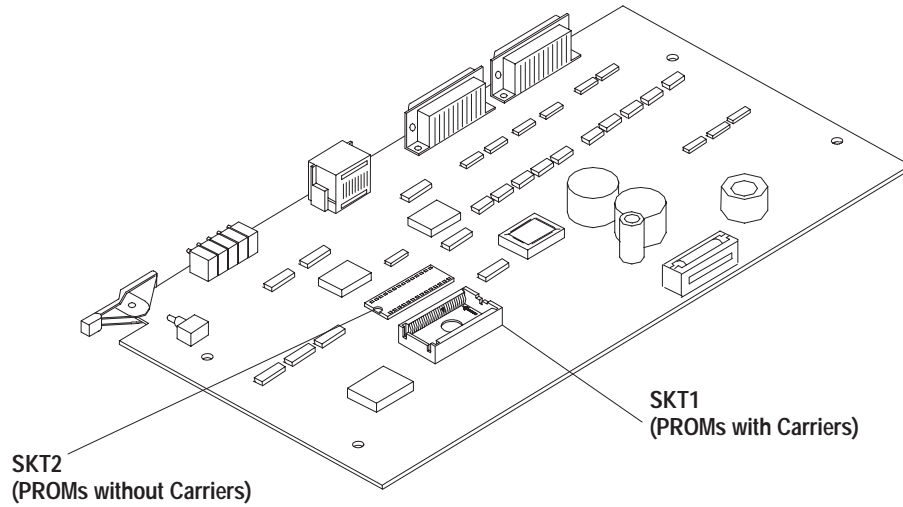
Catalog number	Size	Carrier	Socket	JW2: PROM circuitry	JW3: CPU speed
1771-DBMEM1 1771-DBMEM2	8K byte EEPROM 32K byte EEPROM	yes yes	SKT1 SKT1	 top	 either turbo shown for optimum performance
1747-M1 1747-M2 1747-M3	8K byte EEPROM 32K byte EEPROM 8K byte EPROM	yes yes yes	SKT1 SKT1 SKT1	 top	 normal
1747-M4	32K byte EPROM	yes	SKT1	 bottom	 normal
PN 940654-02 PN 940654-03	32K byte EPROM 32K byte EPROM	no no	SKT2 SKT2	 bottom	 normal

1. Unpack the memory module and check the contents of the package. If any items are missing, contact your local Allen-Bradley office.
2. Remove the BASIC module from the I/O chassis. If you need further instructions to complete this step see page 3 -2.
3. Disassemble the BASIC module to gain access to the memory module sockets. If you need further instructions to complete this step, see page 3 -3.



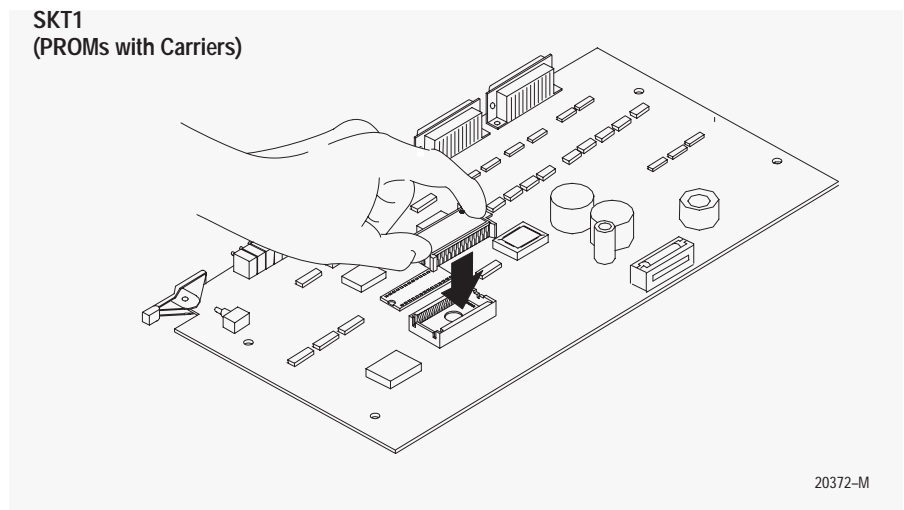
ATTENTION: Electrostatic discharge can damage integrated circuits or semiconductors in the module of you touch backplane connector pins. Wear a wrist-strap grounding device and use a static-safe workstation, if available.

4. Place the board on a flat surface.



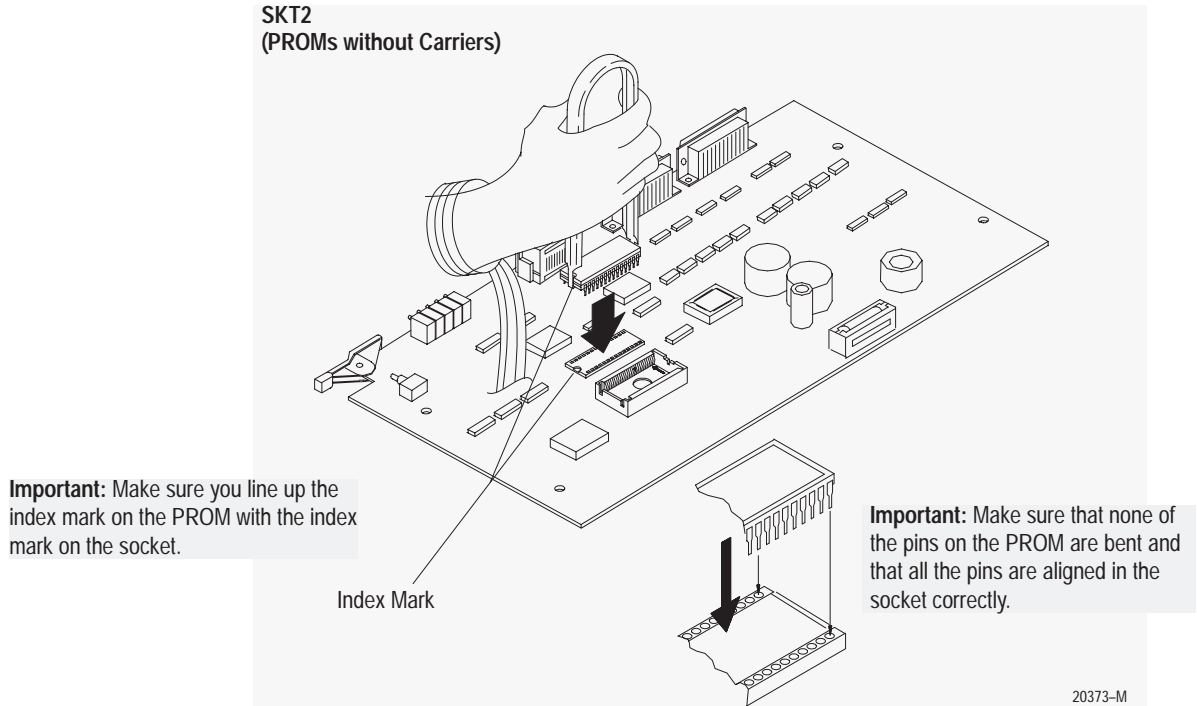
20371-M

5. Insert the memory module into the appropriate socket.

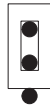



20372-M

Use a chip insertion tool with memory modules that have no carrier.



6. Reassemble the BASIC module. If you need further instructions to complete this step, see page 3 -11.
7. Set jumper JW2 to configure the BASIC module for the circuitry of your memory module. If you have an Allen-Bradley memory module refer to the table on page 3 -5. If not, refer to the Memory Module Configuration Jumper on page 1 -4.
8. Set jumper JW3 (CPU Speed Select Jumper, page 1 -5) to set the speed of your BASIC module CPU. If you have an Allen-Bradley memory module refer to the table on page 3 -5. If not set jumper JW3 according to the speed of your memory module:

Memory speed	JW3 setting
150 ns – 91 ns	 Normal
90 ns or faster	 Turbo
slower than 150 ns	not allowed

9. Replace the BASIC module into the I/O chassis. If you need further instructions to complete this step, see page 1 -12.

Install the Battery

The battery backs up 24K bytes of user RAM and the clock/calendar. Drain on the battery should be less than 0.5 mA dc during battery back-up (no power) and less than 50 uA while the module is powered. Battery life during no-power conditions is about 2,000 hours. Battery shelf life is about 20,000 hours.

When the BTLO LED indicator light comes on the battery should maintain the clock and program data for about three days. We recommend immediate replacement.

The BASIC module ships with a MAXELL ER3STC battery. You can replace it with this battery or a Allen-Bradley battery, catalog number 1770-XZ or a Tadiran battery, part number 15-51-03-210-000.

Important: The BASIC module retains the current configuration for 36 minutes after removing the battery. If it takes you longer than 36 minutes to replace the battery, you lose the program stored in RAM.



ATTENTION: Do not incinerate or dispose lithium batteries in general trash collection. They may explode or rupture violently. Check state and local regulations dealing with the disposal of these materials. You are legally responsible for hazards created while your battery is being disposed.



Ship or dispose the battery according to the recommended procedures listed in the disposal section of *Guidelines for Handling Lithium Batteries* (publication number AG-5.4).

Use CALL 80 (page 13 -9) to monitor battery status.

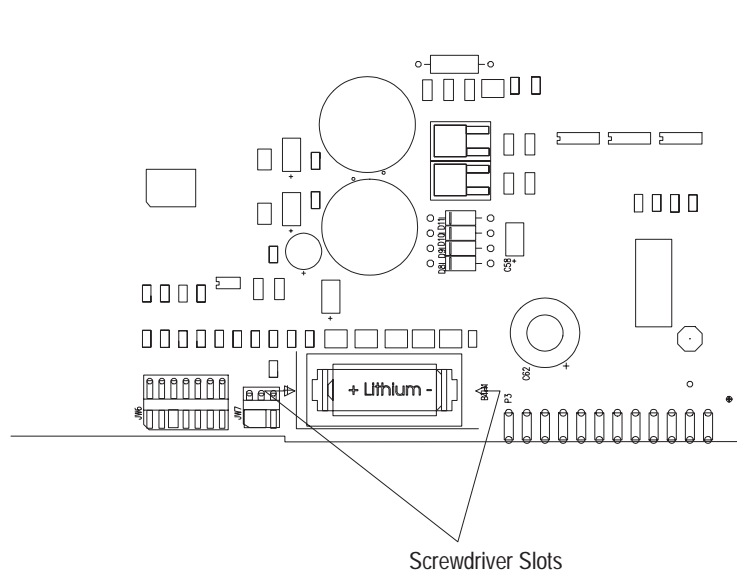
To replace the battery:

1. Remove the BASIC module from the I/O chassis. If you need further instructions to complete this step see page 3 -2.
2. Disassemble the BASIC module. If you need further instructions to complete this step, see page 3 -3.

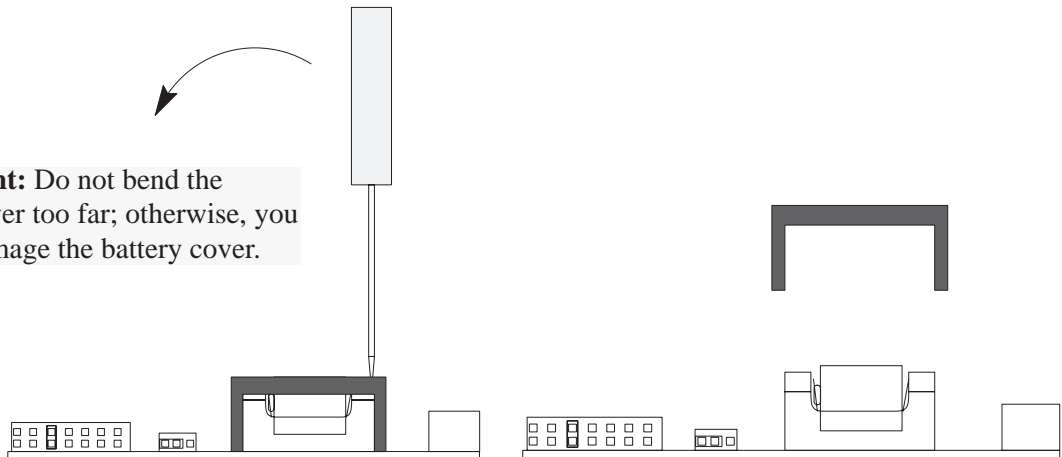


ATTENTION: Electrostatic discharge can damage integrated circuits or semiconductors in the module if you touch backplane connector pins. Wear a wrist-strap grounding device and use a static-safe workstation, if available.

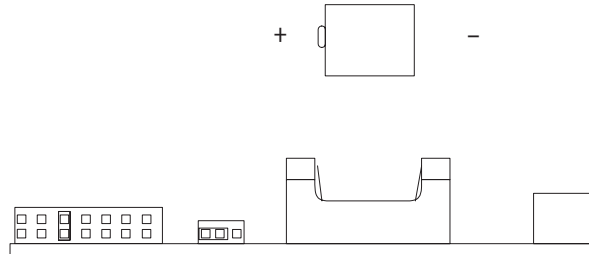
3. Remove the battery cover using a small, flat-blade screwdriver.



Important: Do not bend the screwdriver too far; otherwise, you could damage the battery cover.



4. Remove the battery and insert a fresh one. Make sure you install the battery in the correct orientation.



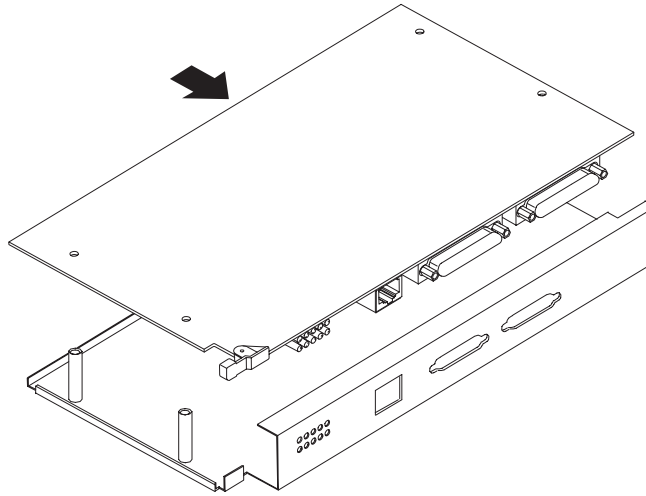
5. Replace the battery cover.
6. Reassemble the BASIC module. If you need further instructions to complete this step, see page 3 -11.
7. Make sure jumper JW7 is enabled (page 1 -9).
8. Replace the BASIC module into the I/O chassis. If you need further instructions to complete this step, see page 1 -12.

The BTLO LED indicator light should go out.

Reassemble the BASIC Module

After installing the memory module or battery, reassemble the BASIC module as follows:

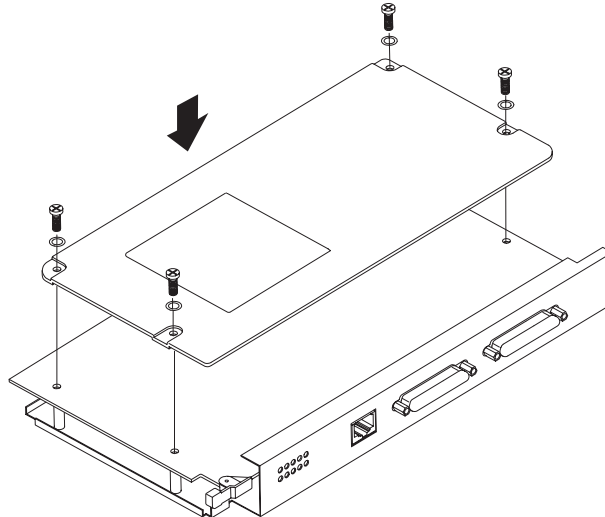
1. Insert the board into the main case.



20370-M

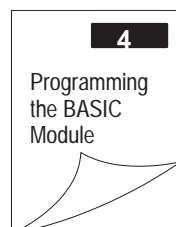
2. Replace the right-side cover.

Important: Make sure you replace the washers. Also, do not over-tighten the screws; otherwise you can damage the cover.



20369-M

What's Next?



- Programming Instructions
- Creating a Program
- Numbering Program Lines
- Entering a Program
- Running/Stopping Program

Notes:

Programming the BASIC Module

What's in This Chapter?

This chapter describes:	On page:
programming instructions	4 -1
create a program	4 -2
number program lines	4 -6
enter a program	4 -7
run and stop a program	4 -9
what's next?	4 -9

Programming Instructions

BASIC programs are composed of BASIC programming instructions grouped together. These instructions are a combination of operators, commands, statements, and system subroutines (CALLs).

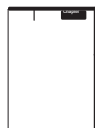
Important: The BASIC module operates in two modes: the Command mode (Program mode) and the Run mode (Interpreter mode). You can only enter commands when the processor is in the Command mode.

BASIC Operators



BASIC operators are programming instructions that you execute during Run mode. You typically use these operators to perform a predefined operation on either variables or constants. Operators require either one or two operands. Chapter 9 describes operators in detail.

BASIC Commands



BASIC commands are programming instructions that initiate an action from the BASIC module. You execute commands in Command mode from the command line. You use these commands to perform some type of program maintenance. Chapter 10 describes these commands in detail.

BASIC Statements



BASIC statements are programming instructions that control program flow or manipulate I/O and memory. Every statement begins with a line number, followed by a statement body, and terminated with a carriage return (CR) or a colon (:) in case of multiple statements per line number. You execute statements automatically within a BASIC program during Run mode. You can also enter these statements from the command line in Command mode to test/evaluate the execution of the statement. Chapter 11 describes these statements in detail. Chapter 7 gives you an overview of how to use the different types of statements.

BASIC Subroutines (CALLs)



BASIC system subroutines (calls) are programming instructions that you execute within a BASIC program or from the command line. A call is actually a type of statement. There are 128 calls. These calls perform such activities as setting the clock/calendar, manipulating strings, processing block transfers, and converting backplane data. Calls 0–127 are described in detail in Chapters 12 and 13. Chapter 7 gives you an overview of how to use the different types of calls.

Create a Program



BASIC module execution is controlled through a BASIC program residing in RAM or ROM (memory module).

Remember these hints as you are programming:

- Define strings first, unless you are executing a CALL 77 (page 13 -6). Then, execute CALL 77 first and define your strings immediately after.
- Dimension arrays after defining strings.
- Define most used variables first. Use 0 until you assign real values.
- When doing math, save intermediate values rather than recalculate.
- Place the most used subroutines near the beginning of the program.
- Straight through code executes faster, but uses more memory.
- Put multiple statements on a line after debugging the program.
- Comments use memory space and slow execution time. After debugging the program save a fully commented copy on disk and remove comments from the executable program.

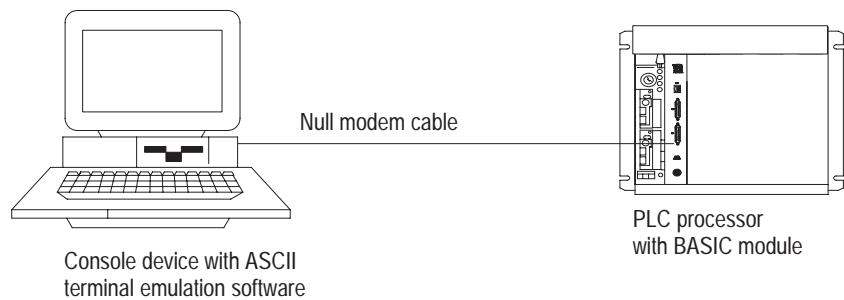
You can create and edit your BASIC program using a personal computer along with BASIC Development Software (PBASE) or using an ASCII terminal or a personal computer running an ASCII terminal emulation software package.

Important: You can also program the BASIC module using the C programming language. Contact your local Allen-Bradley sales office for additional information.

ASCII Terminal Emulator

Use an ASCII terminal to enter a BASIC program one line at a time to your BASIC module through port PRT1 (configured as the program port). The ASCII terminal connected to the BASIC module must be an industrial terminal, workstation, or personal computer (without the BASIC development software) that communicates in alphanumeric mode. An ASCII terminal can also be used to display charts or graphs generated by your BASIC program.

In this configuration, you connect the RS-232 port on the back of your industrial terminal or personal computer to port PRT1 on your BASIC module. Port PRT1 must be configured as the program port.



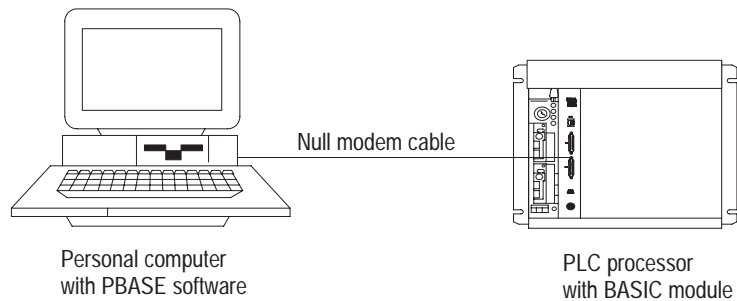
Refer to Chapter 2 for additional information on port configuration.

BASIC Development Software (PBASE)

Use a personal computer with the BASIC Development software (PBASE) to create a BASIC program that is then downloaded to your BASIC module. PBASE provides a structured and efficient means of programming your BASIC module. This software is loaded into a 100% IBM compatible personal computer. It uses the personal computer to facilitate editing, compiling (translating), uploading, and downloading BASIC programs to the BASIC module. You can use PBASE with either the RS-232 or the DH-485 interface. You must use PBASE software when the DH485 port is the program port.

RS-232 Interface

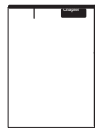
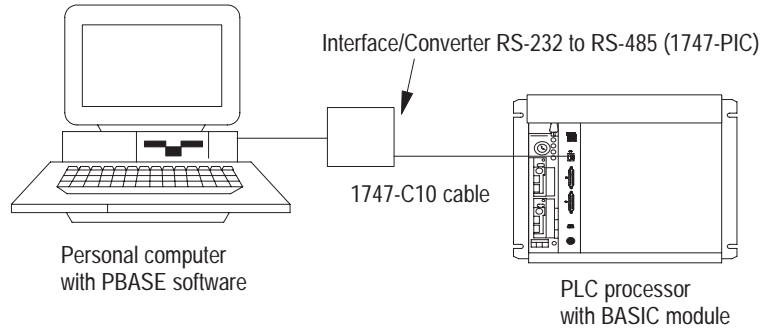
In this configuration, you connect the serial port on the personal computer to port PRT1 on the BASIC module. The personal computer communicates with the BASIC module through terminal emulation over an RS-232 interface. Port PRT1 is configured as the program port.



Refer to Chapter 2 for additional information on port configuration.

DH-485 Interface

In this configuration, you interface the serial port on the personal computer with port DH485 on the BASIC module through a 1747-PIC Interface/Converter. The 1747-PIC Interface/Converter converts the RS-232 signals from the personal computer RS-232 serial port to RS-485 format. Port DH485 is configured as the program port.



Refer to Chapter 2 for additional information on port configuration.



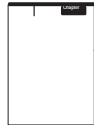
Important: This chapter focuses on using an ASCII terminal to program the BASIC module. Refer to the BASIC Development Software Programming Manual (publication number 1746-6.2) for information on programming the BASIC module with PBASE.

Number Program Lines

BASIC program lines always begin with a line number ranging from 0 to 65535. The line numbers indicate the order in which the program lines are stored in memory. You also use them as references when branching and editing. Typically you start numbering BASIC programs with line number 10 and increment by 10. This allows you to add additional lines later as you work on your program. You can use a line number only once in a program. Each line can contain no more than 79 characters.

Since the computer runs the statements in numerical order, additional lines need not appear in consecutive order on the screen. For example, if you enter line 35 after line 40, the computer still runs line 35 after line 30 and before line 40. This technique saves you from re-entering an entire program if you forget to include a line.

Important: If you reuse an existing line number all information you referenced with original line number is lost. Be careful when entering line numbers in Command mode, you may accidentally erase program lines.



After the line number, you may have a combination of BASIC statements (Chapter 11), operators (Chapter 9), or CALLs (Chapters 12 and 13). Depending on the logic of your program, you may have more than one statement on a line. If so, you must separate each statement with a colon (:).

The BASIC line must contain at least one character after the line number, but no more than 79 characters, including the line number.

Enter a Program



To enter a BASIC program using an ASCII terminal follow these steps.



Important: Refer to the BASIC Development Software Programming Manual (publication number 1746-6.2) for information on entering a program using PBASE software.

1. Select the program port using jumper JW4 (page 1 -6).
2. Connect the ASCII terminal to the BASIC module program port (see page 2 -2).
3. Verify that the console device is configured to communicate with the BASIC module.
4. Apply power to your system.

If a program is not in RAM, this screen appears:

```
BASIC Module - Catalog Number 1771-DB/B  
Firmware Revision: A  
Allen-Bradley Company, Copyright 1989, 1990, 1991, 1992, 1993, 1994  
All rights reserved  
>
```

If a program is in RAM and you programmed the BASIC module to execute from RAM, the program starts running on power up.

Type  +  to stop the program. This screen appears:

```
.  
. .  
STOP - IN LINE XXX  
READY  
>
```

Important: The system prompt > indicates that the BASIC module is now in Command mode.

5. Enter a line of the BASIC program at the system prompt > .

BASIC ignores spaces and automatically inserts them during a LIST command. You can enter lowercase characters in Command mode. However, any key words, commands, statements, variable and array names entered in lowercase change to uppercase when you store the program in memory.

```
READY  
>10 REM FIRST PROGRAM  
>20 PRINT "HELLO WORLD"
```

6. Press  to end the program line.

Run and Stop a Program





Run the Program



To run a BASIC program, type `RUN` (page 10 -19) at the system prompt [`>`].

```
READY  
>RUN  
  
HELLO WORLD  
  
READY  
>
```

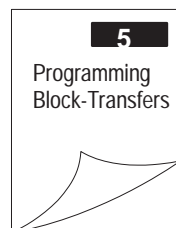
Stop the Program

To stop a running program, press  + .

`CALL 19` (page 12 -12) disables  +  and `CALL 18` (page 12 -11) re-enables  + .

Important: If  +  is disabled, you cannot stop program execution through a BASIC command. In this case, set jumper JW4 (Chapter 1) of the BASIC module in the factory default position and cycle power to stop program execution.

What's Next?



- BASIC Module Memory Organization
- Data Tables
- Block-Transfer Buffers
- Block-Transfers and the BASIC module
- PLC-2 Family Processors Ladder Logic
- PLC-3 Family Processors Ladder Logic
- PLC-5 Family Processors Ladder Logic
- PLC-5/250 Family Processors Ladder Logic

Notes:

Programming Block-Transfers

What's in This Chapter?

This chapter describes:	On page:
BASIC module memory organization	5 -1
data tables	5 -2
block-transfer buffers	5 -4
block-transfers and the BASIC module	5 -5
PLC-2 family processors ladder logic	5 -8
PLC-3 family processors ladder logic	5 -9
PLC-5 family processors ladder logic	5 -10
PLC-5/250 family processors ladder logic	5 -12
what's next?	5 -12

BASIC Module Memory Organization

All data transferred from the PLC to the BASIC module must be routed through the BASIC module **input buffer**. The block transfer write buffer (BTW) is part of the input buffer. This table lists the defined offsets of the BASIC module input buffer. These offset numbers are used by the various calls that manipulate the BTW buffer or the DH-485 common interface file.

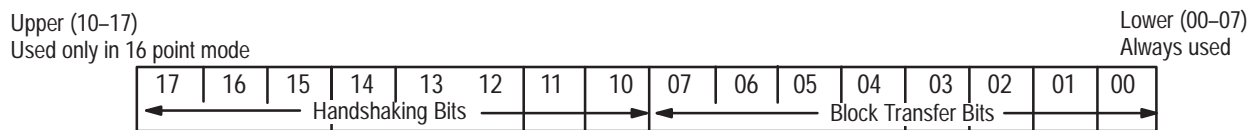
Offset	Definition
0	reserved
1-64	PLC block transfer write from PLC processor
65-99	reserved
100-139	data transferred from the DH-485 common interface file

All data transferred from the BASIC module to the PLC must be routed through the BASIC module **output buffer**. The block transfer read buffer (BTR) is part of the output buffer. This table lists the defined offsets of the BASIC module output buffer. These offsets are used by the various calls that manipulate the BTR buffer or the DH-485 common interface file.

Offset	Definition
0	reserved
1-64	block transfer read to PLC processor
65-99	reserved
100-139	data transferred to the DH-485 common interface file

Data Tables

The BASIC module communicates with any PLC processor that has block-transfer capability. Your ladder logic program and BASIC program work together to enable proper communications between the BASIC module and PLC processor. The BASIC module can perform both block and discrete transfers. Use JW5 (page 1 -7) to set the BASIC module backplane configuration. With JW5 set to 8-point mode, the BASIC module uses 8 bits in both the input and output image table for block transfer. When you have JW5 configured for 16-point mode the firmware also allows you to examine bits 10–17 for handshaking.



Output Image Table

Output Image Table Bit	Description	Used with CALL ^①
00–07	Do not use—Reserved for Block Transfers	–
10	Reserved	–
11	PRT1 BTW Req	34 (PRT1)
12	DF1 BTR Req	122
13	PRT2/DF1 BTW Req	34 (PRT2) 123
14	DH-485 READ Req	49
15	DH-485 WRITE Req	50
16	Module Interrupt Req	32
17	Reserved	–

^①See Chapter 12 and 13 for information on these calls.

Input Image Table

Input Image Table Bit	Description	Used with CALL ^①
00	Do not use-Reserved	-
01	BTW Req	3 and 6
02	BTR Req	2 and 7
03	Do not use-Reserved	-
04	DF1 Status BTR Req	123
05-07	Do not use-Reserved	-
10	PRT1 BTW Req	33 (PRT1)
11	PRT1 BTW Ack	34 (PRT1)
12	PRT2 BTR Req/ DF1 BTR Ack	33 (PRT2), 122
13	PRT2/DF1 BTW Ack	34 (PRT2), 123
14	DH-485 READ Ack	49
15	DH-485 WRITE Ack	50
16	DH-485 Status BTR Req	50
17	Unsolicited DH-485/DF1 WRITE	118

^①See Chapter 12 and 13 for information on these calls.



Important: When the BASIC module is in 16-point mode you must perform all block-transfers synchronously with a PLC-5 family processor. Asynchronous block transfers do not work with this configuration and cause the backplane circuit within the module to lock up. The BASIC module ships with JW5 configured for 8-point mode (Series A compatible). Unless you are using calls 32, 33, 34, 49, 50, 118, 122, or 123 (see Chapters 12 and 13) you should be in 8 point mode.

Important: You cannot use 2-slot chassis addressing if JW5 is set for 16 point mode. See page 1 -10 for information regarding the backplane configuration and addressing.

Block-Transfer Buffers

Block-Transfer Write Buffer

The BASIC module processor maintains a block-transfer-write (BTW) buffer containing the values of the last BTW sent by the PLC processor. Use CALL 4 to set the BTW word length. Transfer data to the BASIC module's BTW buffer with CALL 6 or CALL 3.

Block-Transfer Read Buffer

The BASIC module also maintains a block-transfer-read (BTR) buffer that is the value of the next block, read by the PLC processor. Use CALL 5 to set the BTR word length. Transfer the data to the BASIC module processor BTR buffer (for subsequent transfer to the PLC processor) with CALL 7 or CALL 2. You should complete the building of the read buffer before initiating its transfer.

Important: Use CALL 4 and CALL 5 once, immediately after power-up if possible. CALL 4 sets the BTW block length. CALL 5 sets the BTR block length. Failure to follow these guidelines could cause random lockup of the BASIC module program.

Block-Transfers and the BASIC Module

The BASIC module is a bi-directional block-transfer module. Bi-directional means that the module performs both block-transfer-read and block-transfer-write operations:

- Use a BTR instruction to transfer data (1 to 64, 16-bit words) from your module to the PLC processor's data table.
- Use a BTW instruction to transfer data (1 to 64, 16-bit words) to your module from the PLC processor's data table.

The PLC processor sends variable length blocks of data to the module. You can transfer a maximum of 64 words in and 64 words out per scan. The module responds with the requested block length of data. The module has an auxiliary processor dedicated to servicing of the block-transfers to and from the PLC processor. Use these support routines to provide the communications link between the PLC processor and BASIC module.

Statement		Page
CALL 2	timed-block-transfer-read buffer	12 -2
CALL 3	timed-block-transfer-write buffer	12 -3
CALL 4	set block-transfer-write length	12 -4
CALL 5	set block-transfer-read length	12 -4
CALL 6	block-transfer-write buffer	12 -5
CALL 7	block-transfer-read buffer	12 -5
CALL 120	clear BASIC module input and output buffers	13 -57

Important: Alternate BTR and BTW instructions in ladder logic and between CALL 7 or CALL 2 and CALL 6 or CALL 3 in your BASIC program. BTR and BTW instruction enable bits must not be set at the same time. If you do not, a block-transfer lockup between the PLC processor and BASIC module may occur.

Block-Transfer Programming Tips



Tip

Remember these block-transfer programming tips:

- Block lengths PUSHed for CALLs 4 and 5 must equal the corresponding lengths on your BTW/BTR instructions in the PLC ladder logic.
- If a BTW appears first in your ladder logic, put a CALL 3 or 6 first in your BASIC program. If a BTR appears first in your ladder logic, put a CALL 2 or 7 first in your BASIC program.
- If your application requires bi-directional block-transfers, you can use CALL 6 or 3 and CALL 7 or 2 anywhere in your program at any time interval, if you:
 - Use an equal number of each type.
 - Alternate their use (e.g. CALL 6, CALL 7, CALL 6, CALL 7).
- If your application requires a one way block-transfer (all write-block-transfers or all read-block-transfers), use only the associated CALLs (4 and 3 or 6; 5 and 2 or 7).

Sample BASIC Block-Transfer Program

This sample program assumes that the application requires a single block-transfer-read (BTR) and a single block-transfer-write (BTW) to pass data between the processor and the BASIC module (transfer of 64 words or less). If the transferred data exceeds 64 words, you must program multiple file to file moves to move different data sets to and from the block-transfer files.

The values shown are for demonstration purposes only. Use the program for all PLC-2, PLC-3, PLC-5 and PLC-5/250 processor ladder logic programs shown in this chapter.

```
>5   DIM A(5)
>10  REM SET BTW LENGTH TO 5 WORDS
>20  PUSH 5: CALL 4
>30  REM SET BTR LENGTH TO 5 WORDS
>40  PUSH 5: CALL 5
>50  REM READ THE BTW BUFFER
>60  CALL 6
>70  REM CONVERT DATA FROM 3-DIGIT BCD TO DB FORMAT
>80  FOR I=1 TO 5
>90  PUSH (I): CALL 10: POP A(I)
>95  PRINT A(I),
>100 NEXT I
>110 REM DO A CALCULATION
>120 T=A(1)+A(2)+A(3)+A(4)+A(5):V=T/5
>125 PRINT "AVE=", V
>130 REM CONVERT DATA FROM DB FORMAT TO 3-DIGIT BCD
>140 PUSH T: PUSH 1: CALL 20
>150 PUSH V: PUSH 2: CALL 20
>160 REM WRITE TO THE BTR BUFFER
>170 CALL 7
>180 REM CONTINUE TO BLOCK TRANSFER
>190 GOTO 60
>200 END
```

Important: Use CALL 4 and CALL 5 only once, immediately after power-up, if possible. Failure to observe this guideline could cause random BASIC module program lockups.

PLC-2® Family Processors
Ladder Logic

The Mini-PLC-2 (cat. no. 1772-LN3) and PLC 2/20 (cat. no. 1772-LP1, -LP2) processors use multiple GET instructions to perform block-transfers. Refer to the processor user’s manual for an explanation of multiple GET block-transfers.

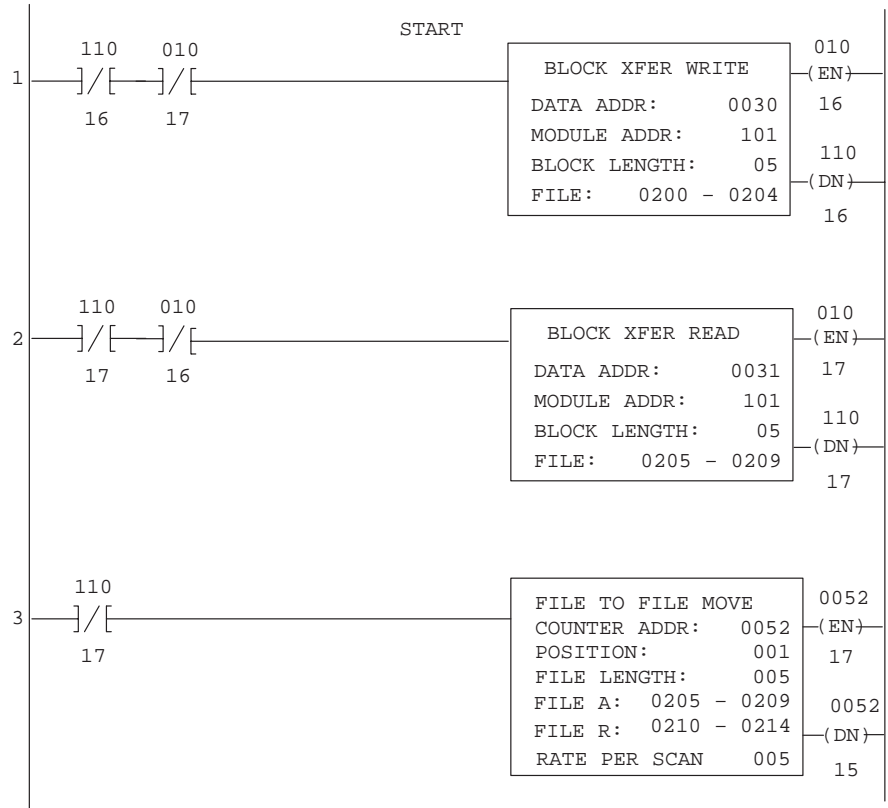
The first two rungs of the sample program toggle the requests for block-transfer-writes (BTW) and block-transfer-reads (BTR). The interlocks shown do not allow a BTR and BTW instruction to enable at the same time.

In Rung 3 when a BTR is successfully completed, its done bit sets, enabling the file-to-file move instruction. The file-to-file move instruction (FFM) moves the BTR data file (File 205–209) into a storage data file (210–214). This prevents the programmable controller from using invalid data if a block-transfer communication fault occurs.

Important: Use the first available timer-counter as a data address.

Important: The PLC-2 is in 2-slot chassis addressing.

LADDER DIAGRAM DUMP



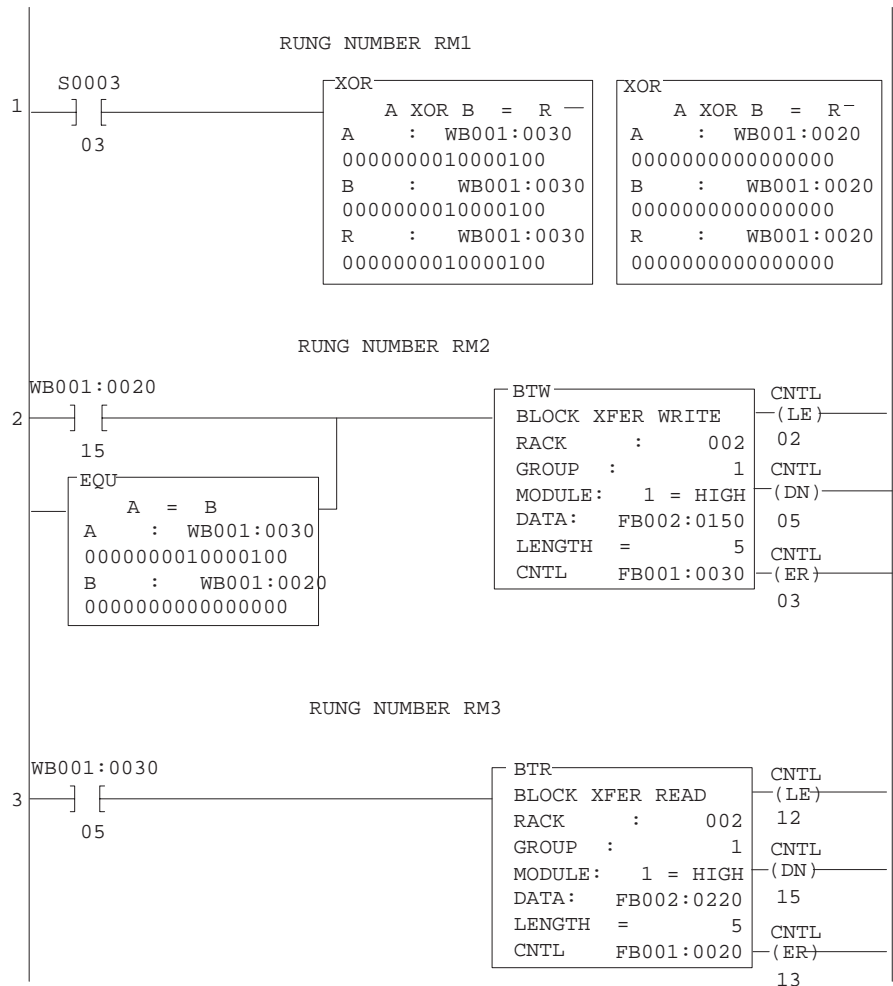
END 01295

15044

PLC-3[®] Family Processors
Ladder Logic

You can use this ladder logic program with PLC-3 or PLC-3/10 processors. This program assumes that your application requires a single BTR and BTW to pass data between the processor and the BASIC module (transfer of 64 words or less). If the transferred data exceeds 64 words you must program multiple file to file moves to move different data sets to and from the block-transfer files.

Rung one is true only at power-up. It uses status word 3, bit 3 (the ac power loss bit of the PLC-3) to zero the control file of both the BTR and BTW. In Rungs 2 and 3, during normal program execution the BTW and BTR instructions are alternately executed. The done bits of each instruction enable the next block-transfer instruction. The BTR and BTW control files must be different for the next block-transfer to occur. The equal instruction is used at power-up. At power-up the BTR and BTW control files both equal zero. At power-up the BTW enables and block-transfers begin.



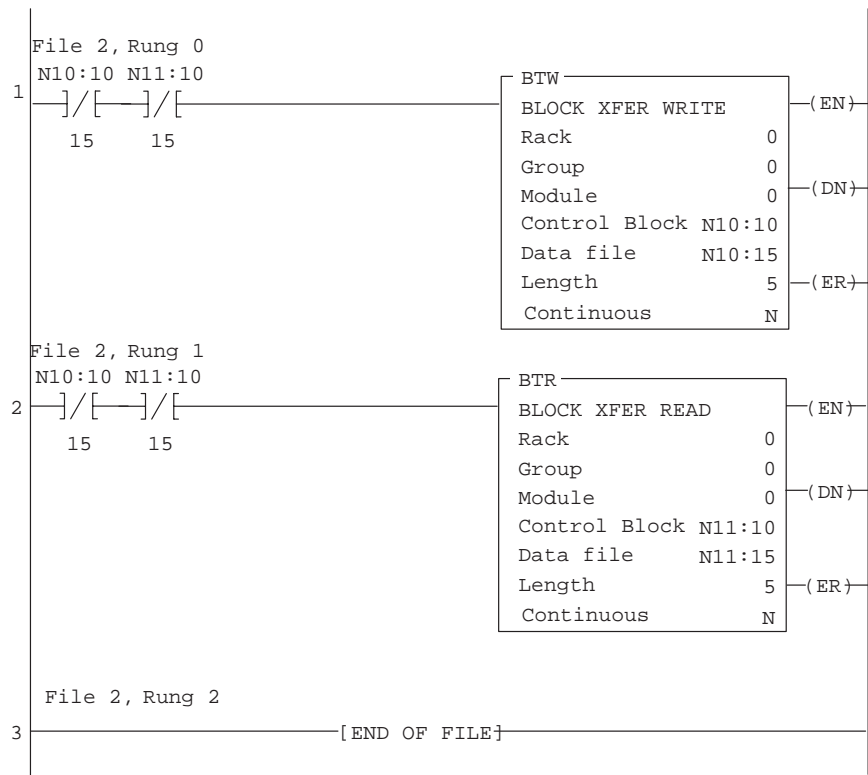
PLC-5[®] Family Processors
Ladder Logic

Asynchronous Block Transfer

You can use this ladder logic program with PLC-5 processors for asynchronous block transfer. This program assumes that your application requires a single block-transfer-read (BTR) and block-transfer-write (BTW) to pass data between the processor and the BASIC module (transfer of 64 words or less). If transferred data exceeds 64 words you must program multiple file-to-file moves to move different data sets to and from block-transfer files.

Rungs 1 and 2 execute the BTR and BTW instructions alternately. When BTR is completed, BTW enables immediately following BTR scan.

Important: Do not select the continuous mode when using bi-directional block-transfer. Continuous mode does not allow use of the status bits in the block-transfer instructions.



15048

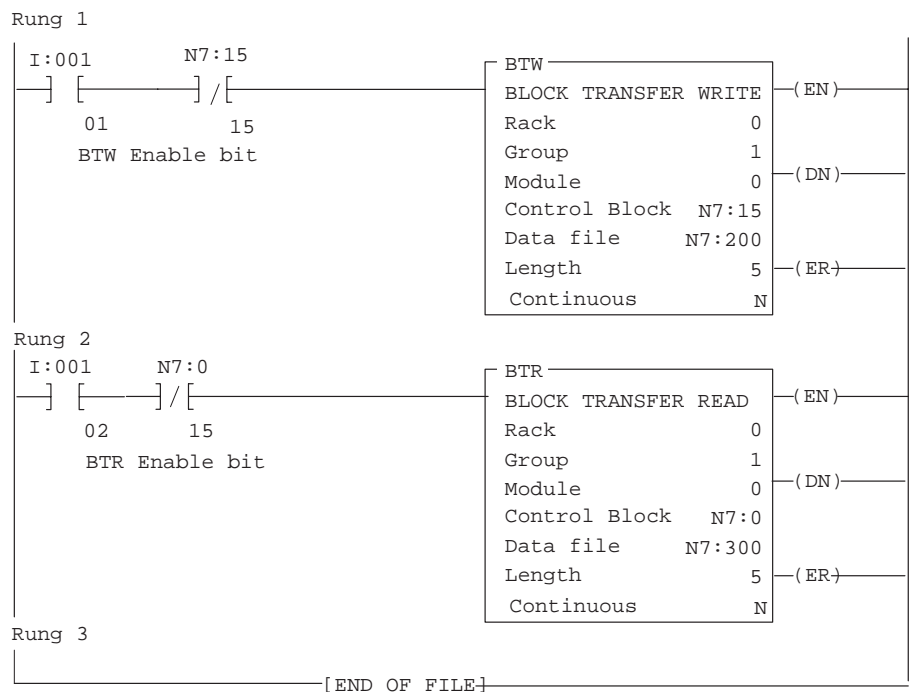
Synchronous Block Transfer

You can use this ladder logic program only with PLC-5 processors to perform synchronous block transfers. This program assumes that your application requires a single block-transfer-read (BTR) and block-transfer-write (BTW) to pass data between the PLC-5 processor and the BASIC module (transfer of 64 words or less). If transferred data exceeds 64 words you must program multiple file-to-file moves to move different data sets to and from block-transfer files.

This example assumes that the BASIC module is in 8 point mode and resides in group 1, slot 1. Rung 1 is an example of a synchronous block-transfer-write. When the BASIC module executes a BTW using CALL 3 or CALL 6, the BASIC module sets bit 01 in the input image byte. Bit 01 enables the BTW instruction in the ladder logic. When the block transfer is complete, the BASIC module resets bit 01.

Rung 2 is an example of a synchronous block-transfer-read. When the BASIC module executes a BTR using CALL 2 or CALL 7, the BASIC module sets bit 02 in the input image byte. Bit 02 enables the BTR instruction in the ladder logic. When the block transfer is complete the BASIC module resets bit 02.

By using this “handshaking” process, the ladder logic enables the block transfers only when the BASIC module requests them. This process should help increase the efficiency of the PLC-5 ladder logic programs when accessing the BASIC module.

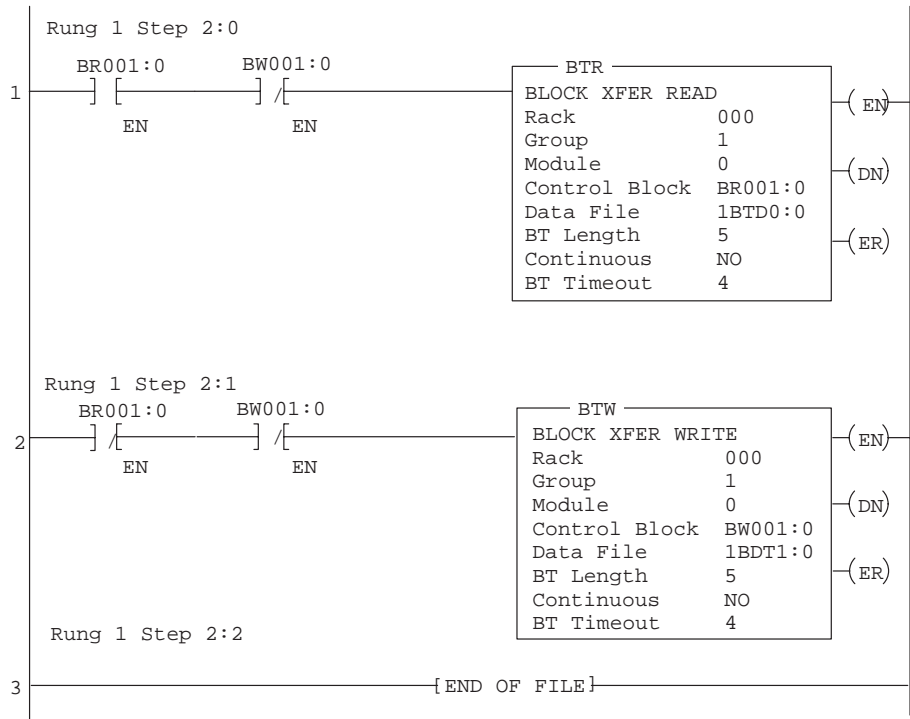


PLC-5/250® Family Processors
Ladder Logic

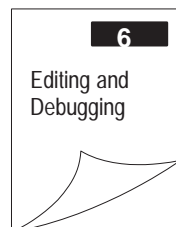
You can use this ladder logic program with PLC-5/250 Family processors. This program assumes that your application requires a single block-transfer-read (BTR) and block-transfer-write (BTW) to pass data between the processor and the BASIC module (transfer of 64 words or less). If the transferred data exceeds 64 words you must program multiple file-to-file moves to move different data sets to and from the block-transfer files.

The first two rungs of the sample program toggle the requests for block-transfer-reads (BTR) and block-transfer-writes (BTW). When the BTW is completed, the BTR enables immediately following the BTW scan. The interlocks shown do not allow a BTR and BTW instruction to enable at the same time.

When a BTW is successfully completed, its done bit sets. A COP instruction can move the BTR data file into a storage file.



What's Next?



- Edit a Program Line
- Delete a Program Line
- Renumber a Program
- Debug a Program

Editing and Debugging a BASIC Program

What's in This Chapter?

This chapter describes:	On page:
edit a program line	6 -1
delete a program line	6 -3
renumber a program	6 -3
debug a program	6 -4
what's next?	6 -4

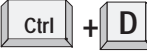

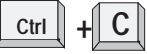
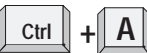

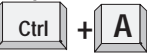



Edit a Program Line

When the BASIC module is in Command mode, you can edit the BASIC program that resides in RAM. Editing a BASIC program is done on a line-by-line basis. To edit an existing line in the BASIC program, type **EDIT** and the line number of the line you want to edit. The ASCII terminal displays the BASIC program line you specify in the EDIT command (page 10 -7).


```
READY
>EDIT 20

20 PRINT "HELLO WORLD"
```

You can perform any of these edit operations:





Operation	Function	Key Strokes
delete	use the delete operation to delete the character at the cursor position	
exit	use the exit operation(s) to exit the editor with or without saving the changes	 - exits the editor and replaces the old line with the edited line  - exits the editor without saving any changes made to the line
insert	use the insert operation to insert text at the current cursor position	
	<p>Important: When you use the insert operation, all text to the right of the cursor disappears until you press the second  command.</p> <p>total line length is 79 characters.</p>	<p>Important: You must press a second  to terminate the insert command</p>
move	use the move operation to provide right/left cursor control	 - moves the cursor one space to the right  - moves the cursor one space to the left
replace	use the replace operation to replace the character at the current cursor position	press the key that corresponds to the character that replaces the character at the current cursor position
retype	use the retype operation to copy the current line of text and insert it at the line following the current line. The cursor moves to the first character on the new line	

Delete a Program Line

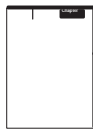
When the BASIC module is in Command mode, you can delete an existing line of the BASIC program. To delete an existing line of the BASIC program, type the line number of the line to delete and press .

Renumber a Program

When the BASIC module is in Command mode, you can renumber the BASIC program that resides in RAM. To renumber a BASIC program, enter a REN command (see page 10-16) at the system prompt >.

This command variation:	Rennumbers the program:
REN 	from the beginning of the program. New line numbers begin at 10 and increment by 10.
REN NUM 	from the beginning of the program. New line numbers begin at 10 and increment by NUM.
REN NUM1, NUM2 	from the beginning of the program. New line numbers begin with NUM1 and increment by NUM2.
REN NUM1, NUM2, NUM3 	starting at NUM2. New line numbers begin with NUM1 and increment by NUM3.

Important:



- REN command updates the destination of GOSUB, GOTO, ONERR, ONTIME and ON GOTO statements (Chapter 11).
- If the target line number does not exist, or if there is insufficient memory to complete the task, no lines are changed and the message `RENUMBER ERROR` appears on the console screen.
- Because the REN command uses the same RAM for renumbering as it does for variable and program storage, available RAM may be insufficient in large programs. Renumber your program periodically and in segments during development.

Debug a Program

The BRKPNT (page 10 -2) command and SNGLSTP command (page 10 -20), along with the STOP statement (page 11 -36) help you to debug your program.

Set Break Points

Use the BRKPNT command to set a program break point at the line number you specify with this command. Program execution stops just before the line number you specified. If the line number is zero, the break point is disabled. After the break point is reached, you can examine variables by using PRINT statements. You can also modify the variables by assignment statements. Continue from the break point by using the CONT command (page 10 -3.) Once the break point is reached, it is disabled. To stop at the same place twice, set the break point twice. The BRKPNT command works only on programs executing from RAM. It does not stop a program executing from ROM.

Initiate Single Step Execution

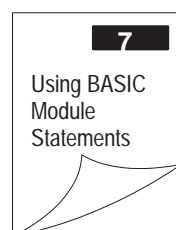
Use the SNGLSTP command to initiate single-step program execution. If the number you specify with this command is zero, single-step execution is disabled. If the number is not zero, a break point is set before each line in the program. Start the program with the RUN command (page 10 -19). After each stop, type CONT (page 10 -3) to execute the next line. You can inspect variables or assign variables at each break point. SNGLSTP works only on programs executing from RAM. It does not stop a program executing from ROM.

Stop Program Execution at a Specific Point

Use the STOP statement to break program execution at specific points in a program. After a program is stopped you can display or modify variables. You can resume program execution with a CONT command (page 10 -3).

Note that the line number printed out after execution of the STOP statement is the line number following the STOP statement, not the line number that contains the STOP statement.

What's Next?



Using BASIC Module Statements

What's in This Chapter?



This chapter groups the statements and calls required to manipulate the various hardware parts of the BASIC module. We assume you are familiar with standard BASIC programming practices. Therefore, standard BASIC commands and statements are not covered here unless a special consideration is required for the BASIC module. Some calls perform multiple operations and are listed in more than one place. Chapter 11 describes the statements in detail and Chapters 12 and 13 describe the calls in detail.

This chapter describes:	On page:
memory and operation calls	7 -1
port communication calls	7 -2
block-transfer support calls	7 -3
number conversion calls	7 -4
clock/calendar calls	7 -4
string calls	7 -5
DH-485 communication	7 -5
DF1 communication	7 -6
background operations	7 -6
command line calls	7 -7
execution control and interrupt support calls	7 -7
input calls	7 -8
output calls	7 -9
setup calls	7 -9
status calls	7 -10
what's next?	7 -10

Memory and Operation Calls

Memory Manipulation

Statement	Page
CALL 70 ROM to RAM program transfer	13 -2
CALL 71 ROM/RAM to ROM program transfer	13 -3
CALL 72 RAM/ROM return	13 -4
CALL 73 battery-backed RAM disable	13 -5
CALL 74 battery-backed RAM enable	13 -5
CALL 77 protected variable storage	13 -6
CALL 80 check battery condition	13 -9
CALL 81 user PROM check and description	13 -10
CALL 82 check user memory module map	13 -11

Miscellaneous

Statement		Page
CALL 18	re-enable control C break function	12 -11
CALL 19	disable the control C break function	12 -12
CALL 32	enable/disable processor interrupt	12 -22
CALL 38	expanded ONERR restart	12 -36
CALL 99	reset print head pointer	13 -34
CALL 109	print the argument stack	13 -44
CALL 112	user LED control	13 -47
CALL 120	clear BASIC module input and output buffers	13 -57

Port Communication Calls

Program Port

Statement		Page
PRINT		11 -29
GET		11 -12
INPL		11 -16
INPS		11 -16
INPUT		11 -17
EOF		9 -17
LIST		10 -9
CALL 78	set program port baud rate	13 -8

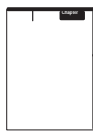
PRT1 Port

Statement		Page
MODE		11 -20
PRINT@		11 -29
GET@		11 -12
INPL@		11 -16
INPS@		11 -16
INPUT@		11 -17
EOF@		9 -17
LIST@		10 -9
CALL 94	display current PRT1 port setup	13 -32
CALL 95	get number of characters in PRT1 buffers	13 -32
CALL 96	clear PRT1 receive/transmit buffers	13 -33
CALL 99	reset print head pointer	13 -34
CALL 103	print PRT1 transmit buffer and pointer	13 -36
CALL 104	print PRT1 receive buffer and pointer	13 -37
CALL 105	reset PRT1 to default settings	13 -37

PRT2 Port

Statement	Page
MODE	11 -20
PRINT#	11 -29
GET#	11 -12
INPL#	11 -16
INPS#	11 -16
INPUT#	11 -17
EOF#	9 -17
LIST#	10 -9
CALL 30	PRT2 port support parameter set 12 -20
CALL 31	display PRT2 port parameters 12 -21
CALL 35	retrieve numeric input character from PRT2 port 12 -34
CALL 36	get number of characters in PRT2 port buffers 12 -35
CALL 37	clear PRT2 port buffers 12 -35
CALL 97	enable PRT2 DTR 13 -33
CALL 98	disable PRT2 DTR 13 -34
CALL 99	reset print head pointer 13 -34
CALL 110	print the PRT2 port transmit buffer and pointer 13 -45
CALL 111	print the PRT2 receive buffer and pointer 13 -46
CALL 119	reset the PRT2 port to default settings 13 -56

Block-Transfer Support Calls



The BASIC module communicates with the PLC processor using block-transfer communications. The PLC processor sends variable length blocks of data to the module. You can transfer a maximum of 64 words in and 64 words out per scan. The module responds with the requested block length of data. The module has an auxiliary processor dedicated to servicing of the block-transfers to and from the PLC processor. Use these support routines to provide the communications link between the PLC processor and the BASIC processor. See Chapter 5 for more information on block-transfers.

Statement	Page
CALL 2	timed-block-transfer-read buffer 12 -2
CALL 3	timed-block-transfer-write buffer 12 -3
CALL 4	set block-transfer-write length 12 -4
CALL 5	set block-transfer-read length 12 -4
CALL 6	block-transfer-write buffer 12 -5
CALL 7	block-transfer-read buffer 12 -5
CALL 120	clear BASIC module input and output buffers 13 -57

Number Conversion Calls



Use these calls to convert numbers between integer and BASIC floating-point. Use these calls also to transfer data to the BASIC module block-transfer-read buffer for transfer to the PLC processor (using CALL 2, page 12 -2 or CALL 7, page 12 -5) and to retrieve data from the BASIC module block-transfer-write buffer after transfer from the PLC processor (using CALL 3, page 12 -3 or CALL 6, page 12 -5). You can use these calls within the BASIC program or from the command line. See Chapter 8 for more information on number conversions.

Statement		Page
CALL 10	3-digit decimal BCD to BASIC floating point	12 -6
CALL 11	16-bit binary to BASIC floating point	12 -7
CALL 12	4-digit octal to BASIC floating point	12 -7
CALL 13	6-digit decimal BCD to BASIC floating point	12 -8
CALL 14	SLC 16-bit signed integer to BASIC floating point	12 -8
CALL 15	SLC 16-bit unsigned integer to BASIC floating point	12 -9
CALL 17	4-digit BCD to BASIC floating point	12 -11
CALL 20	BASIC floating point to 3-digit decimal BCD	12 -12
CALL 21	BASIC floating point to 16-bit binary	12 -13
CALL 22	BASIC floating point to 4-digit octal	12 -13
CALL 23	BASIC floating point to 6-digit decimal BCD	12 -14
CALL 24	BASIC floating point to SLC 16-bit signed integer	12 -15
CALL 25	BASIC floating point to SLC 16-bit binary	12 -16
CALL 26	BASIC floating point to 3.3-digit BCD	12 -17
CALL 27	BASIC floating point to 4-digit BCD	12 -17
CALL 39	3.3-Digit Signed, BCD to BASIC Floating Point	12 -38
CALL 88	BASIC floating point to PLC-5 floating point	13 -16
CALL 89	PLC-5 floating point to BASIC floating point	13 -17

Clock/Calendar Calls

Use these calls to set and display the real time clock/calendar within the BASIC module or from the command line.

Statement		Page
CLOCK 0		11 -4
CLOCK 1		11 -5
ONTIME		11 -25
TIME		9 -19
CALL 40	set wall clock time	12 -39
CALL 41	set wall clock date	12 -40
CALL 42	set wall clock day of week	12 -40
CALL 43	date/time retrieve string	12 -41
CALL 44	date retrieve numeric	12 -41
CALL 45	time retrieve string	12 -42
CALL 46	time retrieve numeric	12 -42
CALL 47	retrieve day of week string	12 -43
CALL 48	retrieve day of week numeric	12 -43
CALL 52	date retrieve string	12 -58

String Calls

Use these calls to manipulate string data structures within a BASIC program or from the command line.

Statement	Page
STRING	11 -37
CALL 60 string repeat	12 -59
CALL 61 string append	12 -60
CALL 62 number to string conversion	12 -61
CALL 63 string to number conversion	12 -62
CALL 64 find a string in a string	12 -63
CALL 65 replace a string in a string	12 -64
CALL 66 insert a string in a string	12 -65
CALL 67 delete a string from a string	12 -66
CALL 68 determine length of a string	12 -67

DH-485 Communication

Use these calls when the DH485 port is configured for DH-485 communications.

Important: CALLs 29, 49, 50, and 118 are for background operation. Do not attempt to execute standard DH-485 calls while background calls are enabled and active. Invalid data transfers could result.

Statement	Page
CALL 29 read/write to PLC/SLC from module internal string	12 -18
CALL 49 read remote DH-485 SLC data file	12 -44
CALL 50 write to remote DH-485 SLC data file	12 -50
CALL 83 display DH485 port setup	13 -11
CALL 84 transfer DH-485 CIF to BASIC input buffer	13 -12
CALL 85 transfer BASIC output buffer to DH-485 CIF file	13 -13
CALL 86 check DH-485 interface remote write status	13 -14
CALL 87 check DH-485 interface file remote read status	13 -15
CALL 90 read remote DH-485 data file to BASIC input buffer	13 -18
CALL 91 write BASIC output buffer to remote DH-485 data file	13 -22
CALL 92 read remote DH-485 CIF to BASIC input buffer	13 -26
CALL 93 write output buffer to remote DH-485 CIF file	13 -29
CALL 118 PLC/SLC unsolicited writes	13 -52
CALL 120 clear BASIC module input and output buffers	13 -57

DF1 Protocol Communication

Use these calls when you have PRT2 port configured for DF1 protocol.

Important: CALL 108 must be used before any standard or background DF1 calls.

Important: CALLs 29, 118, 122 and 123 are for background operation. Do not attempt to execute standard DF1 calls while background calls are enabled and active. Invalid data transfers could result.

Statement	Page
ONDF1	11 -22
CALL 16 enable/disable DF1 packet interrupt	12 -10
CALL 29 read/write to PLC/SLC from module internal string	12 -18
CALL 108 enable DF1 driver communications	13 -38
CALL 113 disable DF1 driver communications	13 -47
CALL 114 transmit DF1 packet	13 -48
CALL 115 check DF1 status	13 -49
CALL 117 get DF1 packet length	13 -51
CALL 118 PLC/SLC unsolicited writes	13 -52
CALL 120 clear BASIC module input and output buffers	13 -57
CALL 122 read remote DF1 PLC data file	13 -58
CALL 123 write to remote DF1 PLC data file	13 -66

Background Operations

Use these calls to perform background operations while the BASIC module is executing a program.

Statement	Page
CALL 16 enable/disable DF1 packet interrupt	12 -10
CALL 33 transfer data from PRT1/PRT2 to BTR buffer	12 -23
CALL 34 transfer data from BTW buffer to PRT1/PRT2	12 -29
CALL 49 read remote DH-485 SLC data file	12 -44
CALL 50 write to remote DH-485 SLC data file	12 -50
CALL 118 PLC/SLC unsolicited writes	13 -52
CALL 122 read remote DF1 PLC data file	13 -58
CALL 123 write to remote DF1 PLC data file	13 -66

Command Line Calls

Use these calls to cause a function to occur within the BASIC module. You cannot execute these calls within the BASIC program, but rather enter them at the command line.

Statement		Page
CALL 31	display PRT2 port parameters	12 -21
CALL 73	battery-backed RAM disable	13 -5
CALL 74	battery-backed RAM enable	13 -5
CALL 77	protected variable storage	13 -6
CALL 81	user PROM check and description	13 -10
CALL 82	check user memory module map	13 -11
CALL 83	display DH485 port setup	13 -11
CALL 94	display current PRT1 port setup	13 -32
CALL 101	upload user (E)EPROM code to host	13 -35
CALL 103	print PRT1 transmit buffer and pointer	13 -36
CALL 104	print PRT1 receive buffer and pointer	13 -37
CALL 109	print the argument stack	13 -44
CALL 110	print the PRT2 port transmit buffer and pointer	13 -45
CALL 111	print the PRT2 receive buffer and pointer	13 -46

Execution Control and Interrupt Support Calls

Use these calls to control data flow and program transfer between ROM and RAM within the BASIC program.

Statement		Page
GOSUB		11 -13
ONDF1		11 -22
ONERR		11 -23
ON-GOSUB		11 -24
ONTIME		11 -25
POP		11 -28
PUSH		11 -30
RETI		11 -33
RETURN		11 -34
STOP		11 -36
CALL 16	enable/disable DF1 packet interrupt	12 -10
CALL 32	enable/disable processor interrupt	12 -22
CALL 38	expanded ONERR restart	12 -36
CALL 70	ROM to RAM program transfer	13 -2
CALL 71	ROM/RAM to ROM program transfer	13 -3
CALL 72	RAM/ROM return	13 -4

Input Calls

Use these calls to allow the BASIC module to read input data from its external ports. You can execute these calls within a program or from the command line.

Statement		Page
GET		11 -12
INPL		11 -16
INPS		11 -16
INPUT		11 -17
LD@		11 -18
READ		11 -31
CALL 29	read/write to PLC/SLC from module internal string	12 -18
CALL 33	transfer data from PRT1/PRT2 to BTR buffer	12 -23
CALL 35	retrieve numeric input character from PRT2 port	12 -34
CALL 36	get number of characters in PRT2 port buffers	12 -35
CALL 84	transfer DH-485 CIF to BASIC input buffer	13 -12
CALL 90	read remote DH-485 data file to BASIC input buffer	13 -18
CALL 92	read remote DH-485 CIF to BASIC input buffer	13 -26
CALL 95	get number of characters in PRT1 buffers	13 -32
CALL 117	get DF1 packet length	13 -51
CALL 118	PLC/SLC unsolicited writes	13 -52
CALL 122	read remote DF1 PLC data file	13 -58

Output Calls

Use these calls to allow the transfer of data from the BASIC module to external ports PRT1, PRT2, and DH485 within the BASIC program or from the command line.

Statement		Page
PH0		11 -27
PH1		11 -27
PRINT		11 -29
ST@		11 -35
CALL 29	read/write to PLC/SLC from module internal string	12 -18
CALL 31	display PRT2 port parameters	12 -21
CALL 34	transfer data from BTW buffer to PRT1/PRT2	12 -29
CALL 37	clear PRT2 port buffers	12 -35
CALL 49	read remote DH-485 SLC data file	12 -44
CALL 50	write to remote DH-485 SLC data file	12 -50
CALL 83	display DH485 port setup	13 -11
CALL 85	transfer BASIC output buffer to DH-485 CIF file	13 -13
CALL 91	write BASIC output buffer to remote DH-485 data file	13 -22
CALL 93	write output buffer to remote DH-485 CIF file	13 -29
CALL 94	display current PRT1 port setup	13 -32
CALL 96	clear PRT1 receive/transmit buffers	13 -33
CALL 100	download/program assembly language to EEPROM	13 -35
CALL 112	user LED control	13 -47
CALL 114	transmit DF1 packet	13 -48
CALL 116	call user defined assembly language routine	13 -50
CALL 123	write to remote DF1 PLC data file	13 -66

Setup Calls

Use these calls to set port parameters within a BASIC program or from the command line.

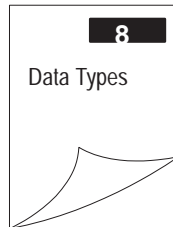
Statement		Page
MODE		11 -20
CALL 18	re-enable control C break function	12 -11
CALL 19	disable the control C break function	12 -12
CALL 30	PRT2 port support parameter set	12 -20
CALL 78	set program port baud rate	13 -8
CALL 99	reset print head pointer	13 -34
CALL 105	reset PRT1 to default settings	13 -37
CALL 108	enable DF1 driver communications	13 -38
CALL 113	disable DF1 driver communications	13 -47
CALL 119	reset the PRT2 port to default settings	13 -56

Status Calls

Use these calls to monitor the status of the BASIC module. You can execute these calls from a BASIC program or from the command line.

Statement		Page
CALL 80	check battery condition	13 -9
CALL 86	check DH-485 interface remote write status	13 -14
CALL 87	check DH-485 interface file remote read status	13 -15
CALL 115	check DF1 status	13 -49

What's Next?



- Argument Stack
- Control Stack
- String Data Types
- Numeric Data Types
- Backplane Conversion Data Types

Data Types

What's in This Chapter?

This chapter describes:	On page:
argument stack	8 -1
control stack	8 -1
string data types	8 -2
numeric data types	8 -3
backplane conversion data types	8 -4
what's next?	8 -9

Argument Stack

The argument stack (A-stack) stores all constants that the BASIC module is currently using. Operations (see Chapter 9) such as add, subtract, multiply, and divide always operate on the first two numbers of the argument stack and return the result to the stack. The argument stack is 203 bytes long. The BASIC module stores all constants and variables as floating point. Each floating point number placed in the stack requires 6 bytes of storage. The argument stack can hold up to 33 floating point numbers before overflowing.

In addition, the PUSH command (page 11 -30) saves data to the argument stack and the POP (page 11 -28) command restores data from the stack. PUSHes and POPs are typically associated with call routines. Pushes and Pops are mechanisms you use to transfer information to and from call routines. PUSH makes a copy of the variable you PUSHed, then puts that copy on the top of the argument stack. POP takes the value on the top of the argument stack and copies it to the variable you POPped.

Control Stack

The control stack (C-stack) stores all information associated with loop control (ex. DO-WHILE, DO-UNTIL, FOR-NEXT, etc.) The control stack is 157 bytes long. DO-WHILE and DO-UNTIL loops use 3 bytes of the control stack. FOR-NEXT loops use 17 bytes of the the control stack. Calculate the number of operations you are using times the number of bytes each operation uses to determine how many levels you can nest.

For example, if you have no GOSUBs to return from, no DO-UNTIL or FOR-NEXT loops running you can nest DO-WHILE loops 52 levels (157/3). Or if you have no GOSUBs to return from, no DO-UNTIL or DO-WHILE loops running you can nest FOR-NEXT loops 9 levels (157/17). Typically your program uses several different combinations of statements that use the C-stack memory.

String Data Types

A string is a character or group of characters stored in memory. Usually, the characters stored in a string make up a word or a sentence. Strings allow you to use characters instead of numbers. Strings are shown as $\$(expr)$.

The BASIC module uses single-dimension string variables, $\$(expr)$. The dimension of a string variable (the *expr* value) ranges from 0 to 254. This means you can define and manipulate 255 different strings in the BASIC module. Initially, memory is not allocated for strings. You allocate memory with the STRING statement (page 11 -37). Declare and manipulate strings through the \$ type declaration character (page 9 -2).

You can only use one STRING statement in your program to allocate memory for all the strings you want to use in your program. When allocating memory for strings remember that the STRING statement itself has one overhead byte. As well, BASIC uses one overhead byte per string declared within the STRING statement.

For example, `STRING 106,20` allocates 106 bytes for string memory. The 106 bytes of string memory includes five 20-byte strings (100 bytes), five overhead bytes (1 per string) and one additional byte (for the STRING statement itself). The BASIC module automatically numbers these five strings as $\$(0)$, $\$(1)$, $\$(2)$, $\$(3)$, and $\$(4)$.

In the BASIC module you can define strings with the:

- LET statement (page 11 -19)
- INPUT statement (page 11 -17)
- ASC operator (page 9 -14)



Tip

Remember define strings first, unless you are executing a CALL 77 (page 13 -6). Then, execute CALL 77 first and define your strings immediately after.

Numeric Data Types

Two numeric data types exist:

- integer
- floating-point

You can enter and display these numeric data types in four formats:

- integer (ex. 129)
- decimal (ex. 34.98)
- hexadecimal (ex. 0A6EH)
- exponential (ex. 1.23456E+3)

The BASIC module interprets all numbers as floating point numbers except when performing logical operations. When performing logical operations, the BASIC module converts floating point numbers to integers, performs the operation, then converts the result back to floating point.

Integer Numbers

The BASIC module operates on unsigned 16-bit integers, ranging from 0 to 65535 or 0FFFFH. You can enter all integers in either decimal or hexadecimal format. You indicate a hexadecimal number by placing the character H after the number (ex.170H). If the hexadecimal number begins with A - F, then you must precede it by a zero. For example, you must enter A567H as 0A567H. When an operator, such as .AND. (page 9 -7) requires an integer, the BASIC module truncates the fraction portion of the number so it fits the integer format. For example, both 6.3 and 6.8 when interpreted as an integer would equal 6.

Floating-Point Numbers

The BASIC module stores all numbers as floating-point numbers. Floating-point numbers are numbers in which the decimal point floats depending on the significant digits of a specific number. The BASIC module accounts for the location of the decimal point. This allows the BASIC module to store only the significant digits of a value, thus saving memory space.

You can represent $\pm 1E^{-127}$ to $\pm .99999999^{+127}$ in the BASIC module. There are eight significant digits. Numbers are internally rounded to fit this precision.

Backplane Conversion Data Types

The BASIC module communicates with the local processor through the I/O chassis backplane. All data communicated to and from the PLC processor is in PLC format.

The BASIC module interfaces with the PLC-2, PLC-3 and PLC-5 family processors. Converted data is exchanged with programmable controllers using block-transfers.

You can send these data type to the PLC processor:

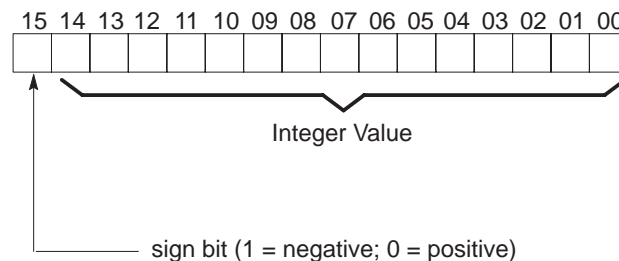
- SLC 16-bit signed integer (-32768 to 32767)
- SLC 16-bit binary (0 to 65, 535)
- 16-bit binary (0000000000000000 to 1111111111111111)
- 3-digit, signed, fixed decimal BCD (\pm XXX.)
- 4-digit, unsigned, fixed decimal BCD (XXXX.)
- 4-digit, signed, octal (\pm XXXX)
- 6-digit, signed fixed decimal BCD (\pm XXXXXX.)
- 3.3-digit, signed, fixed decimal BCD (\pm XXX.XXX)
- floating point ($\pm 1.1754944E^{-38}$ to $\pm 3.4028237E^{+38}$)

You can send these data types to the BASIC module:

- 16-bit signed integer (-32768 to 32767)
- SLC 16-bit unsigned integer (0 to 65,535)
- SLC 16-bit binary (4 Hex digits)(XXXX)
- 3-digit, signed, fixed decimal BCD (\pm XXX.)
- 4-digit, unsigned, fixed decimal BCD (XXXX.)
- 4-digit, signed, octal (\pm XXXX.)
- 6-digit, signed, fixed decimal BCD (\pm XXXXXX.)
- 3.3-digit, signed, fixed decimal BCD (\pm XXX.XXX)
- floating point ($\pm 1.1754944E^{-38}$ to $\pm 3.4028237E^{+38}$)

SLC 16-Bit Signed Integer

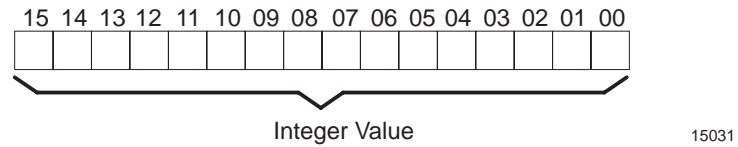
This value requires one word of the processor data table. The data is represented by 15 integer bits and one sign bit, bit 15. The value ranges from -32,768 to 32,767. If you use a value that is not within this range you get a BAD ARGUMENT error and program execution halts. Note that these calls are used with DH-485 calls .See CALL 14 (page 12 -8) and CALL 24 (page 12 -15).



15031

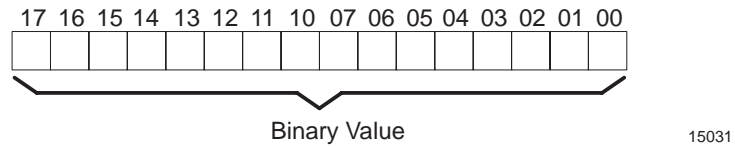
SLC 16-Bit Unsigned Integer (SLC 16-Bit Binary)

This value requires one word of the processor data table. The data is represented by 16 straight binary. The value ranges from 0 to 65,535. If you use value less than 0, then the value placed in the output buffer is 0. If you use a value greater than 65,535 the value placed in the output buffer is 65,535. You are responsible for checking the range of the number before conversion. Note that these calls are used with DH-485 calls. See CALL 15 (page 12 -9) and CALL 25 (page 12 -16).



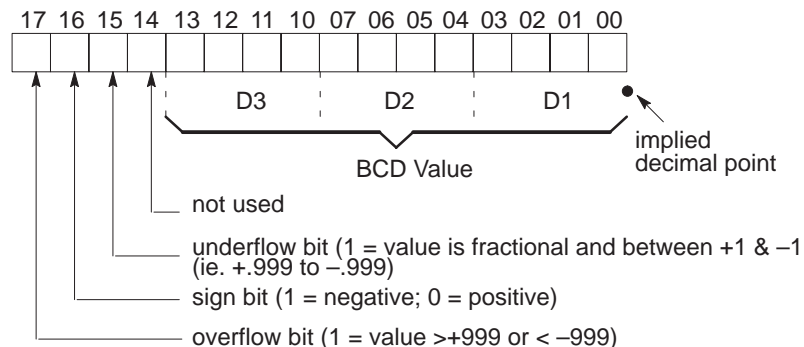
16-Bit Binary (4 Hex Digits)

This value requires one word of the processor data table. The data is represented by 16 straight binary bits. The value ranges from 0 to 65,535. No sign, overflow or underflow bits are affected or decoded. If you use a value larger than 65,535 or a negative number you get a BAD ARGUMENT error and program execution halts. See CALL 11 (page 12 -7) and CALL 21 (page 12 -13),



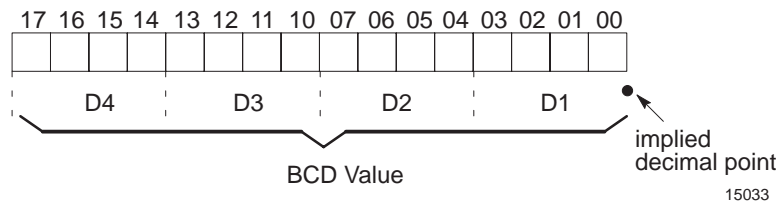
3-Digit, Signed, Fixed Decimal BCD

This value requires one word of the processor data table. The data is represented by a 3-digit binary coded decimal integer. Overflow, underflow and sign are also indicated. An underflow or overflow condition sets the appropriate bit and a value of 000 is returned. The value ranges from -999 to +999. Fractional portions of any number used with the routine are truncated. See CALL 10 (page 12 -6) and CALL 20 (page 12 -12).



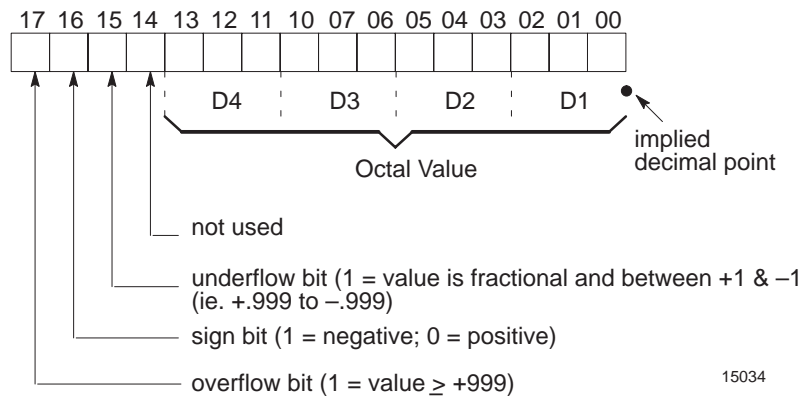
4-Digit, Unsigned, Fixed Decimal BCD

This value requires one word of the processor data table. The data is represented by a 4-digit BCD integer. The value ranges from 0–9999. There is no indication of sign, underflow or overflow. However, if a value of greater than 9999 is converted or an invalid number, the value reported is 0000. Fractional portions of any number used with the routine are truncated. See CALL 17 (page 12 -11) and CALL 27 (page 12 -17).



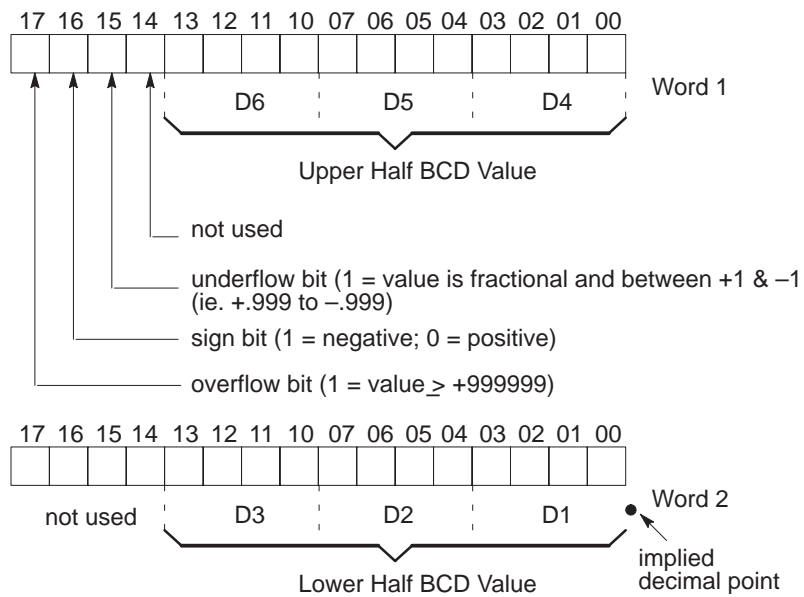
4-Digit, Signed, Octal

This value requires one word of the processor data table. The data is represented by a 4-digit octal integer. The value ranges from $\pm 7777_8$ (± 4095). Overflow, underflow and sign are also indicated. If an overflow or underflow condition exists, the appropriate bit is set and the value of 000 is reported. Fractional portions of any number used in this routine are truncated. See CALL 12 (page 12 -7) and CALL 22 (page 12 -13).



6-Digit, Signed, Fixed Decimal BCD

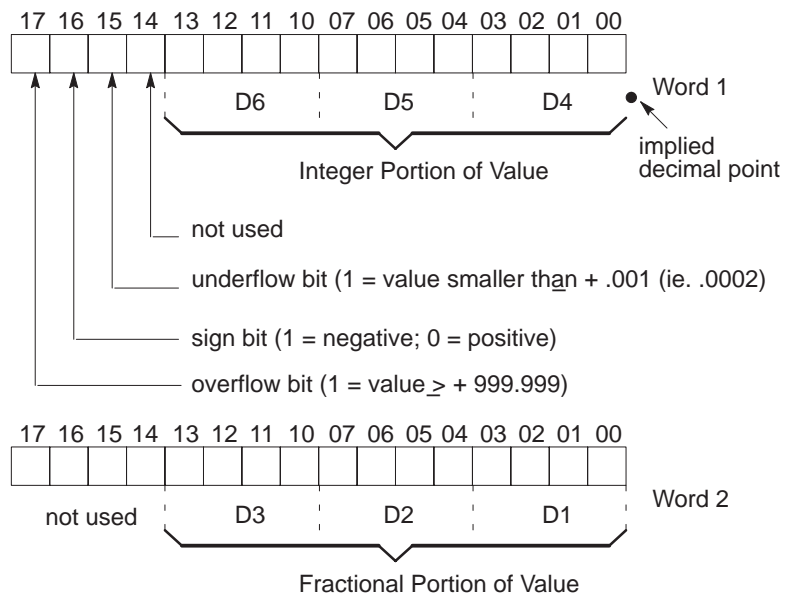
This value requires two words of the processor data table. The first word contains overflow, underflow and sign data and the three most significant digits of the 6-digit BCD integer. The second word contains the lower three digits of the value. The value ranges from -999999 to +999999. If an overflow or underflow condition exists, the appropriate bit is set and a value of 000000 is reported. Fractional portions of any number used with this routine are truncated. See CALL 13 (page 12 -8) and CALL 23 (page 12 -14).



15035

3.3-Digit, Signed, Fixed Decimal BCD

This value requires two words of the processor data table. The first word contains the overflow, underflow, sign data and the three most significant digits of the value. The second word contains the lower three digits of the value. The value ranges from -999.999 to +999.999. If an overflow or underflow condition exists, a value of 000.000 is reported and the appropriate bit is set. Any digits more than 3 places to the right of the decimal point are truncated. See CALL 26 (page 12 -17) and CALL 39 (page 12 -38).



15036

Floating Point

The PLC-5 floating point number is a 7-digit binary floating point number (IEEE Float 32-bit value). The values range from:

$$\pm 1.1754944E^{-38} \text{ to } \pm 3.4028237E^{+38}$$

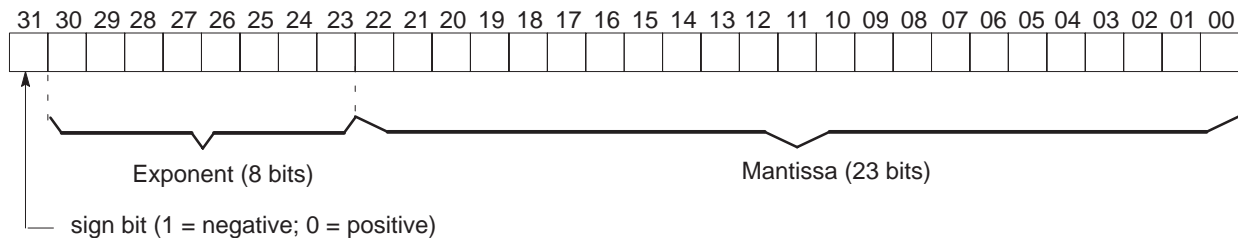
The BASIC module floating point number uses an 8-digit BCD floating point number. The range of the BASIC module floating point number is:

$$\pm 1E^{-127} \text{ to } \pm 99999999E^{+127}$$

The BASIC module has a floating point range larger than the floating point range of the PLC-5 processor. If CALL 88 attempts to convert a number larger than $\pm 3.4028237E^{+38}$, the converted number is assigned a value of $\pm 3.4028237E^{+38}$. If CALL 88 attempts to convert a number smaller than $\pm 1.1754944E^{-38}$, the converted number is assigned a value of $\pm 1.1754944E^{-38}$.

PLC-5 floating point numbers are stored in 2 words of the BTR or BTW buffer. See CALLs 88 and 89 (page 13 -16).

Important: Due to the fact that the PLC-5 floating point number is a 7-digit floating point number, and the BASIC module is an 8-digit floating point number, some round off error may be introduced during number conversions.



What's Next?



Notes:

Expressions, Variables and Operators

What's in This Chapter?

This chapter describes:	On page:
expressions	9 -1
relational expressions	9 -1
constants	9 -1
variables	9 -2
order of operations	9 -3
arithmetic operators	9 -5
bitwise operators	9 -7
relational operators	9 -9
trigonometric operators	9 -10
functional operators	9 -11
logarithmic operators	9 -13
string operators	9 -14
special function operators	9 -17
what's next?	9 -20

Expressions

An expression is a logical mathematical expression that uses operators, constants, and variables to produce a result. Expressions can be simple or complex.

- simple expression: $A+B-C$ or $10*X/20$
- complex expression: $12*(\text{SIN}(A)*\text{SIN}(A)+\text{COS}(A))*(\text{COS}(A)/2)$

A stand alone variable or constant is also considered an expression.

Relational Expressions

Relational expressions involve the operators EQUAL (=), NOT EQUAL (<>), GREATER THAN OR EQUAL TO (>=), and LESS THAN OR EQUAL TO (<=). You use them in control statements to test a condition (ex. IF $A < 100$ THEN...). Relational expressions always require two operands. See relational operators on page 9 -9 for more information.

Constants

A constant is a real number that ranges from $\pm 1E^{-127}$ to $\pm .9999999 9E^{+127}$. A constant can be an integer.

Variables

Variables may represent either numeric values or strings. Numeric values are floating point variables and do not require a type declaration. Strings are string variable types and do require a type declaration. The type declaration character for string is \$. Variable names:

- Must be no more than 8 characters long.
- Must have a unique first and last character when the variable length is the same. (For example, the BASIC module reads “cat” and “cot” as the same variable name. However, “cat” and “cod” are unique and so are “cat” and “chant”.)
- Can include letters, numbers and a decimal point.
- Cannot be a reserved word.
- Can include special type declaration characters (\$ for string)

A variable can be a letter followed by a:

- single-dimension expression (ex. $G(A+6)$)
- number followed by a single-dimension expression (ex. $G2(A+6)$)
- number or letter (ex. $G1$ or GA) except for these combinations:
CR, DO, IE, IF, IP, ON, PI, SP, TO, UI, UO

Important: Embedded reserved words cannot be used as variable names (ex. **FORT**, **PRINTER**, **LENGTH**).

Variables that include a single-dimension expression *expr* are called dimensioned or arrayed variables (ex. $J(4)$, $G(A+6)$). Variables that contain a letter or a letter and a number are called scalar variables (ex. AA , $A1$). Any variables entered in lower case are changed to upper case.

The BASIC module allocates variables in a static manner, meaning the first time a variable is used, the BASIC module allocates a portion of memory (8 bytes) specifically for that variable. You cannot de-allocate this memory on a variable to variable basis. For example, if you execute a statement, you cannot tell the BASIC module that the variable no longer exists to free up the 8 bytes of memory that belong to that variable. You can clear the memory allocated to variables with a CLEAR statement (page 11 -2). The CLEAR statement frees all memory allocated to variables. Variables may be set aside for reuse to save memory.

Important: The BASIC module requires less time to find a scalar variable because there is not an expression to evaluate. To run a program as fast as possible, use dimensioned variables only when necessary. Use scalar variables for intermediate variables and assign the final result to a dimensioned variable. Also, put the most frequently used variables first. Variables defined first require the least amount of time to locate.

Order of Operations

Eight types of operators may act on an expression:

- arithmetic
- logical
- relational
- trigonometric
- functional
- logarithmic
- string
- special function

An operator performs a defined operation on variables or constants. Operators require either one (ex. SIN, COS, and ABS) or two operands (ex. +, -, *, /).

You can write complex expressions using only a small number of parentheses. An expression is scanned from left to right. Operators of the higher precedence are performed before operators of lower precedence. If the operators are of equal precedence they are performed as encountered from left to right. The precedence of operators from highest to lowest:

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.)

To illustrate the order of operations, examine this equation:

$$4+3*2$$

In this equation you cannot perform addition until the multiplication operation is complete because multiplication has a higher precedence. Therefore, multiply (3*2) and then add 4.

$$4+3*2 = 10$$

Important: Use parentheses if you are in doubt about the order of precedence or to enhance program readability.

Operator	Function	Page
ABS	return the absolute value of expression	9 -11
+	add expressions together	9 -5
ASC	return integer value of ASCII character	9 -14
ATN	return arctangent of argument	9 -11
@ or #	communication direction	9 -17
CBY	retrieve data from core or program memory address location	9 -18
CHR	convert numeric expression to ASCII value	9 -16
COS	return the cosine of argument	9 -10
DBY	retrieve/assign data to/from internal data memory	9 -18
/	divide first expression by second expression	9 -5
EOF	test for empty input buffer	9 -17
EXP	raise e to power of argument	9 -13
**	raise first expression by the power of the second expression	9 -5
FREE	list available bytes in RAM	9 -17
INT	return integer portion of expression	9 -12
LEN	list amount of bytes in current program	9 -17
LOG ()	return the natural log of the argument	9 -13
.AND.	combine the first expression with the second expression using .AND.	9 -7
.OR.	combine the first expression with the second expression using .OR.	9 -8
.XOR.	combine the first expression with the second expression using .XOR.	9 -8
MTOP	return last valid memory address	9 -18
*	multiply expressions together	9 -5
-	negation	9 -6
NOT	returns the one's complement or inverse of the number	9 -11
PI	store constant. PI	9 -12
RND	return a pseudo-random number	9 -12
SGN	return the sign of argument	9 -12
SIN	return the sine of argument	9 -10
SQR	return the square root of the argument	9 -12
-	subtract one expression from another	9 -5
TAN	return the tangent of argument	9 -10
TIME	read/assign the free running clock	9 -19
XBY	read/assign external data memory	9 -19
=	allow the first expression to equal the second expression	9 -9
<	allow the first expression to be less than the second expression	9 -9
<=	allow the first expression to be less than or equal to the second expression	9 -9
>	allow the first expression to be greater than the second expression	9 -9
>=	allow the first expression to be greater than or equal to the second expression	9 -9
<>	allows the first expression to be unequal to the second expression	9 -9

Arithmetic Operators

The BASIC module contains a complete set of two-operand and one-operand arithmetic operators.

The general form of all two-operand instructions is:

(expr) OP (expr), where OP is one of these arithmetic operators.

Add (+)

Use the addition operator to add the first and the second expressions together.

```
>PRINT 3+2  
Result: 5
```

Divide (/)

Use the division operator to divide the first expression by the second expression.

```
>PRINT 100/5  
Result: 20
```

Exponentiation (**)

Use the exponentiation operator to raise the first expression to the power of the second expression. The maximum power to which you can raise a number is 255. (See also EXP page 9 -13)

```
>PRINT 2**3  
Result: 8
```

Multiply (*)

Use the multiplication operator to multiply the first expression by the second expression.

```
>PRINT 3*3  
Result: 9
```

Subtract (-)

Use the subtraction operator to subtract the second expression from the first expression.

```
>PRINT 9-6  
Result: 3
```

Negation (-)

Use the negation operator to change an expression from positive to negative.

```
>PRINT -(9+4)  
Result: -13
```

Arithmetic Errors

During the evaluation of an expression if a number is too large (overflow) or too small (underflow), or division by zero error occurs, the BASIC module generates error messages and reverts to Command mode.

The largest result allowed from any calculation is $0.99999999 \text{ E}^{+127}$. If you exceed this number, the BASIC module generates the `ERROR: ARITH. OVERFLOW` message and returns to Command mode.

The smallest result allowed from any calculation is $0.99999999 \text{ E}^{-128}$. If you exceed this number, the BASIC module generates the `ERROR: ARITH. UNDERFLOW` message and returns to Command mode.

If you attempt to divide any number by zero, the BASIC module generates the `ERROR: DIVIDE BY ZERO` message and returns to Command mode.

Refer to the `ONERR` statement (page 11 -23) for more information on how to trap these errors.

Bitwise Operators

The BASIC module contains a complete set of bitwise logical operators. Bitwise operators perform their operations on a bit-by-bit level. Therefore, BASIC changes the integers in the expressions to HEX/binary. BASIC can perform bitwise operations on numbers between 0 (0000H) and 65535 (0FFFFH) inclusive. If the argument is outside this range, then the BASIC module generates an `ERROR: BAD ARGUMENT` message and returns to Command mode. All decimal places are truncated, not rounded.

Refer to this truth table when performing bitwise operations.

When X is:	and Y is:	X.AND.Y bit set to:	X.OR.Y bit set to:	X.XOR.Y bit set to:
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Note: 0 = false/off
X = a bit in the first operand

1 = true/on
Y = a bit in the second operand

.AND.

Use the .AND. operator to perform a bitwise AND on two expressions. The .AND. operator compares the bits of the two expressions and sets the bit to 1 (true) if both bits being compared are set to 1 (true).

```
>PRINT 3.AND.2           Result: 2
```

This example performs a bitwise AND on the integers 3 and 2 and prints the result. First, BASIC converts 3 and 2 to binary:

```
3 = 0011
2 = 0010
```

Then the BASIC module compares each digit of the binary number to determine if both bits are set to a 1. For example, the first digit of both numbers is 0 (false) so the result is 0.

```
0011
0010
0010
```

BASIC converts the resulting binary number back into an integer and prints the result.

```
0010 = 2
```

.OR.

Use the bitwise .OR. operator to perform a bitwise OR on two expressions. The .OR. operator compares the bits of the two expressions and sets the bit to 1 (true) if either of bits being compared is set to 1 (true).

```
>PRINT 1.OR.4           Result: 5
```

This example performs a bitwise OR on the integers 1 and 4 and prints the result. First, BASIC converts 1 and 4 to binary:

```
1 = 0001  
4 = 0100
```

Then the BASIC module compares each digit of the binary number to determine if either of the bits is set to a 1. For example, the last digit of the first operand is 1 (true) and the last digit of the second operand is 0 (false) so the resulting bit is set to 1.

```
0001  
0100  
0101
```

The BASIC module converts the resulting binary number back into an integer and prints the result

```
0101 = 5
```

.XOR.

Use the bitwise .XOR. operator to perform an exclusive OR on two expressions. The .XOR. compares the two bits of the expressions and sets the bit to 1 (true) only if one of the bits in the comparison is set to 1 (true).

```
>PRINT 7.XOR.6         Result: 1
```

This example performs a bitwise XOR on the integers 7 and 6 and prints the result. First, BASIC converts 7 and 6 to binary:

```
7 = 0111  
6 = 0110
```

Then the BASIC module compares each digit of the binary number to determine if only one of the bits is set to a 1. For example, the second digit of first operand is 1 (true) and the second digit of the second operand is 1 (true) so the resulting bit is set to 0 (false) because both digits are set to 1.

```
0111  
0110  
0001
```

The BASIC module converts the resulting binary number back into an integer and prints the result.

```
0001 = 1
```

Relational Operators

Relational expressions involve the operators =, < >, >, >=, <, and <=. In the BASIC module, you typically use relational operations to test a condition.

The BASIC module 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 returns to the argument stack. Because of this, it is possible to display the result of a relational expression.

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

You can chain relational expressions with the logical operators .AND., .OR., and .XOR. (page 9 -8). This makes it possible to test a complex condition with one statement.

```
>10 IF (A>E) .AND. (A>C) .OR. (A>D) THEN . . .
```

Additionally, you can use the NOT operator (page 9 -11).

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

Important: When using logical operators to link relational expressions, you must be sure operations are performed in the proper sequence. When in doubt, use parentheses.

Trigonometric Operators

The BASIC module contains a complete set of trigonometric operators. These operators are one-operand operators. The SIN, COS, and TAN operators use a Taylor series to calculate the function. These operators first reduce the argument to a value between 0 and $\text{PI}/2$. This reduction is accomplished with the equation:

$$\text{reduced argument} = (\text{user arg}/\text{PI} - \text{INT}(\text{user arg}/\text{PI})) * \text{PI}$$

The reduced argument, from the above equation, is between 0 and PI . The reduced argument is then tested to see if it is greater than $\text{PI}/2$. If it is, then it is subtracted from PI to yield the final value. If it is not, then the reduced argument is the final value.

Although this method of angle reduction provides a simple and economical means of generating the appropriate arguments for a Taylor series, there is an accuracy problem associated with this technique. The accuracy problem is noticed when the user argument is large (ex: greater than 1000). This is because significant digits in the decimal (fraction) portion of the reduced argument are lost in the expression shown above.

Important: As a general rule, keep the arguments for the trigonometric functions as small as possible.

SIN

Use the SIN operator to return the sine of the argument. The argument is expressed in radians. Calculations are carried out to 7 significant digits. The argument must be between ± 200000 .

```
>PRINT SIN(PI/4)    Result: .7071067
>PRINT SIN(0)       Result: 0
```

COS

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

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

TAN

Use the TAN operator to return the tangent of the argument. The argument is expressed in radians. The argument must be between ± 200000 .

```
>PRINT TAN(PI/4)    Result: 1
>PRINT TAN(0)       Result: 0
```


ATN

Use the ATN operator to return 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 $\pm\pi/2$ (3.1415926/2).

```
>PRINT ATN(PI)      Result: 1.2626272
>PRINT ATN(1)      Result: .78539804
```

Functional Operators

The BASIC module contains a complete set of functional operators. These operators are single-operand operators.

ABS

Use the ABS operator to return the absolute value of the expression.

```
>PRINT ABS(5)      Result: 5
>PRINT ABS(-5)     Result: 5
```

NOT

Use the NOT operator to return the one's complement of the expression. The one's complement is the inverse of the number. BASIC converts the integer to binary and reverses the state of each bit within the number (ex. it turns 1s to 0s and 0's to 1's). Then BASIC converts the result back into integer format. The expression must be a valid integer (ex. between 0 and 65535 (0FFFFH) inclusive). Non-integers are truncated, not rounded.

```
>PRINT NOT(65000)  Result: 535
```

This example calculates the one's complement for the number 65000 and prints the result. First, BASIC converts 65000 to binary:

```
650000 = 1 1 1 1 1 0 1 1 1 1 0 1 0 0 0
```

Then the BASIC module replaces the 1's with 0's and the 0's with 1s.

```
1 1 1 1 1 0 1 1 1 1 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0 1 0 1 1 1
```

BASIC converts the resulting binary number back into an integer and prints the result.

```
0 0 0 0 0 1 0 0 0 0 1 0 1 1 1 = 535
```

INT

Use the INT operator to return the integer portion of the expression.

```
>PRINT INT(3.7)      Result: 3
>PRINT INT(100.876) Result: 100
```

PI

PI is a stored constant. In the BASIC module PI is stored as 3.1415926.

SGN

Use the SGN (sign) operator to return a value of +1 if the argument is greater than zero, 0 if the argument is equal to zero, and -1 if the argument is less than zero.

```
>PRINT SGN(52)      Result: 1
>PRINT SGN(0)       Result: 0
>PRINT SGN(-8)     Result: -1
```

SQR

Use the SQR operator to return the square root of the argument. The argument may not be less than zero.

```
>PRINT SQR(9)       Result: 3
>PRINT SQR(45)     Result: 6.7082035
>PRINT SQR(100)    Result: 10
```

RND

Use the RND operator to return a pseudo-random number in the range between 0 and 1 inclusive. The RND operator uses a 16-bit binary seed (a constantly changing number used as a starting number for the RND calculation) and generates 65536 pseudo-random numbers before repeating the sequence. The RND operator uses the same calculations to arrive at each number, but uses a different seed each time. The numbers generated are between 0/65535 and 65535/65535 inclusive (ex. 0.1, 0.00009, 0.345689).

```
>PRINT RND          Result: .26771954
```

Logarithmic Operators

The BASIC module contains a complete set of logarithmic operators. These operators are one-operand operators.

LOG

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

```
>PRINT LOG(12)      Result: 2.484906
>PRINT LOG(EXP(1))  Result: 1
```

If you need base 10 log, use this expression:

$$\log_{10}(x)=\log(x)/\log(10)$$

EXP

Use the EXP operator to raise the number e (2.7182818) to the power of the argument. This operator is the inverse of the LOG operator. The EXP operator is similar to the ** operator in that it raises a number to a power. However, it is different from the ** operator in that the only number that is raised is 2.7182818. (See also ** page 9 -5.)

```
>PRINT EXP(1)      Result: 2.7182818
>PRINT EXP(2)      Result: 7.3890559
>PRINT EXP(3)      Result: 20.0855362
>PRINT EXP(LOG(2)) Result: 2
```

String Operators

Two operators in the BASIC module can manipulate strings. These operators are ASC and CHR.

ASC

Use the ASC operator to return the integer value of the ASCII character placed in the parentheses.

The BASIC module capitalizes all ASCII characters. The decimal representation for the ASCII character “A” is 65. The decimal representation for the ASCII character “a” is 97. However the BASIC module prints 65 for both characters.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT ASC(a)
>20 PRINT ASC(A)
READY
>RUN
65
65
READY
>
```

Do not use special ASCII characters with a decimal value greater than 127.

In addition, you can evaluate individual characters in a defined ASCII string with the ASC operator. When you use the ASC operator, the $\$(expr)$ denotes what string you want to access. The expression after the comma selects an individual character in the string. In this example, the first character in the string is selected. The decimal representation for the ASCII character T is 84. String character position 0 is invalid.

```
>5  STRING 1000,40
>10 $(1) ="THIS IS A STRING"
>20 PRINT $(1)
>30 PRINT ASC$(1),1)
>40 END
READY
>RUN
THIS IS A STRING
84
READY
>
```

The numbers printed in this example represent the ASCII characters A through L.

```
>NEW
>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1)="ABCDEFGHIJKL"
>20 FOR X = 1 TO 12
>30 PRINT ASC($(1),X),
>40 NEXT X
>50 END
READY
>RUN
65 66 67 68 69 70 71 72 73 75 74 76
READY
>
```

You can also use the ASC operator to change individual characters in a defined string. In general, the ASC operator lets you manipulate individual characters in a string.

```
>NEW
>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) = "ABCDEFGHIJKL"
>20 PRINT $(1)
>30 ASC($(1),1) = 75 : REM DECIMAL EQUIVALENT OF K
>40 PRINT $(1)
>50 ASC($(1),2) = ASC($(1),3)
>60 PRINT $(1)
READY
>RUN
ABCDEFGHIJKL
KBCDEFGHIJKL
KCCDEFGHIJKL
READY
>
```

CHR

Use the CHR operator to convert a numeric expression to an ASCII character.

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

Like the ASC operator, the CHR operator also selects individual characters in a defined ASCII string. The expressions within the parentheses that follow the CHR operator have the same meaning as the expressions in the ASC operator.

```
>NEW
>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) = "THE BASIC MODULE"
>20 FOR I = 1 TO 16 : PRINT CHR$(1),I,: NEXT I
READY
>RUN
THE BASIC MODULE
READY
>
```

Unlike the ASC operator, you *cannot* assign the CHR operator a value. A statement such as CHR \$(1,1)= H is invalid and generates an `ERROR: BAD SYNTAX` message,. When this error occurs the BASIC module enters Command mode. Use the ASC operator to change a value in a string, or use the string support CALL 65 (page 12 -64) - replace string in a string.

Important: Use CHR only in a PRINT statement (page 11 -29). You cannot use the CHR in a DATA statement (page 11 -6).

Special Function Operators

The BASIC module contains a complete set of special function operators. These operators manipulate the I/O hardware and memory addresses of the BASIC module.

and @

Use the # and @ operators to direct communications. Communication takes place through port PRT1 when you program the @ operator.

```
>10 A = GET@
```

Result: Next character in PRT1 input buffer assigned to variable A.

Communication takes place through port PRT2 when you program the # operator.

```
>10 A = GET#
```

Result: Next character in PRT2 input buffer is assigned to variable A.

The absence of either the # or @ operators indicates that communication should take place through a program port (port PRT1 or port DH485).

EOF

Use the EOF operator to test for an empty input buffer before executing an input statement or function. This prevents input statements from waiting indefinitely on empty input buffers. Use the EOF# statement to test for an empty input buffer for port PRT2. Use the EOF@ statement to test for an empty input buffer for port PRT1.

```
>10 REM EXAMPLE PROGRAM  
>20 IF (NOT(EOF)) THEN A=GET  
>30 REM IF BUFFER NOT EMPTY, READ SINGLE CHARACTER
```

FREE

Use the system control value FREE to tell you how many bytes of RAM are available to you. When the current selected program is in RAM, this relationship is true:

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

LEN

Use the system control value LEN to tell you how many bytes of memory the current program occupies. This is the length of the program and does not include the size of string, variable or array memory. You cannot assign LEN a value, you can only read it. A null program (ex. no program) returns a LEN of 1. The 1 represents the end of program file character.

MTOP

Use the MTOP operator to retrieve the last valid memory address in RAM that is available to the BASIC module. After reset, the BASIC module sizes the external memory and assigns the last valid memory address to the system control value MTOP. The module does not use any external RAM beyond the value assigned to MTOP. If a CALL 77 (page 13 -6) has not changed this value, then the last valid BASIC address is 5FFFH (24575).

```
>PRINT MTOP      Result: 24575  
>PH0.MTOP       Result: 5FFFH
```

Important: Use CALL 77 to change MTOP. Do not use MTOP = 22000.

CBY

Use the CBY operator to retrieve data from the program or code memory address location of the BASIC module. You cannot assign CBY a value; you can only read it. The argument for the CBY operator must be a valid integer between 0 and 65535 (0FFFFH). If it is not a valid integer, a bad argument error occurs.

```
>A = CBY(1000)  
Result: The value in the program or code memory address location  
1000 is assigned to variable A.
```

DBY

Use the DBY operator to retrieve or assign data to or from the internal RAM data memory of the BASIC module. Both the value and the argument in the DBY operator must be between 0 and 255 inclusive. This is because there are only 256 internal memory locations in the BASIC module and one byte can only represent a quantity between 0 and 255 inclusive.

```
>A = DBY(11)  
Result: Value in internal memory location 11 assigned to variable A.  
The internal memory location must be between 0 and 255.
```

```
>DBY(250) = OCH  
Result: Value OCH assigned to internal memory location 250.
```

Important: Improper use of this operator may cause a malfunction of the BASIC module.

Important: The DBY operator cannot access the Special Function Registers (SFR).

XBY

Use the XBY operator to retrieve or assign data to or from the external RAM data memory of the BASIC module. The argument for the XBY operator must be a valid integer between 0 and 65535 (0FFFFH). The value assigned to the XBY operator must be between 0 and 65535 (0 and 0FFFFH) inclusive. If not, a bad argument error occurs.

```
>A = XBY(0F000H)
```

Result: Value in external memory location 0F00H assigned to variable A.

```
>XBY(4000H) = 1FH
```

Result: Value 1FH assigned to external memory location 4000H.

Important: Improper use of this operator may cause a malfunction of the BASIC module.

TIME

Use the TIME operator to retrieve or assign a value to the free running clock resident in the BASIC module. After reset, time is equal to 0. CLOCK1 (page 11 -5) enables the free running clock. When you enable the free running clock, the special function operator TIME increments once every 5 milliseconds. The units of time are in seconds.

When you assign TIME a value with a LET statement (page 11 -19) (ex: TIME=100), only the integer portion of TIME is changed.

```
>CLOCK1 :REM ENABLE FREE RUNNING CLOCK
>CLOCK0 :REM DISABLE FREE RUNNING CLOCK-TIME NOT RESET
>PRINT TIME REM: DISPLAY TIME
READY
>RUN
3.315

>NEW
>TIME = 0 :REM SET TIME = 0
>PRINT TIME :REM DISPLAY TIME
READY
>RUN
.315 (Note: only the integer is changed)
```

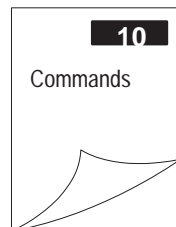
You can change the fraction portion of TIME by manipulating the contents of internal memory location 71 (47H). You can do this by using a DBY(71) operator (page 9 -18). Note that each count in internal memory location 71 (47H) represents 5 milliseconds of TIME.

```
>DBY(71) = 0 :REM FRACTION OF TIME = 0
>PRINT TIME
READY
>RUN
0

>NEW
>DBY(71) = 3 :REM FRACTION OF TIME = 3*5ms = 15 ms
>PRINT TIME
READY
>RUN
1.5 E-2
```

Only the integer portion of TIME changes when a value is assigned. This allows you to generate accurate time intervals. For example, to create an accurate 12-hour (43200 seconds) clock use an ONTIME statement (page 11 -25). When the TIME interrupt occurs, the statement TIME 0 is executed, but the millisecond counter is not re-assigned a value. If interrupt latency exceeds 5 milliseconds, the clock remains accurate.

What's Next?



Commands

What's in This Chapter?




BASIC commands are programming instructions that you execute during the Command mode except for **Ctrl** + **C**. **Ctrl** + **C** takes you from Run mode to Command mode. You typically use these commands to perform some type of program maintenance. You can also execute statements (Chapter 11) and calls (Chapters 12 and 13) from the command line.

This chapter describes:	On page:
BRKPNT	10 -2
CONT	10 -3
CTRL C	10 -4
CTRL Q	10 -5
CTRL S	10 -6
EDIT	10 -7
ERASE	10 -8
LIST	10 -9
NEW	10 -10
NULL	10 -10
PROG	10 -11
PROG1	10 -12
PROG2	10 -13
RAM	10 -15
REN	10 -16
ROM	10 -17
RROM	10 -18
RUN	10 -19
SNGLSTP	10 -20
VER	10 -21
XFER	10 -21
command line calls	10 -22
what's next?	10 -22

BRKPNT

Use the BRKPNT command to set a program break point at the line number you specify with this command. Program execution stops just before the line number you specified. If the line number is zero, the break point is disabled. After the break point is reached, you can examine variables by using PRINT statements. You can also modify the variables by using assignment statements. Continue from the break point by using the CONT command (page 10 -3.) Once the break point is reached, it is disabled. To stop at the same place twice, set the break point twice. The BRKPNT command works only on programs executing from RAM. It does not stop a program executing from ROM.

Syntax





BRKPNT *line number* 

Example

```
>10 D=0 : SU=0 : AV=0
>20 REM GET 100 DATUM POINTS
>30 FOR I=1 TO 100
>40 REM GET ANOTHER DATUM
>50 GOSUB 140
>60 REM SUM THE DATA
>70 GOSUB 170
>80 NEXT I
>90 REM AVERAGE THE DATA
>100 GOSUB 200
>110 REM PRINT RESULT
>120 PRINT "THE AVERAGE VALUE IS ",AV
>130 END

>140 REM THIS SUBROUTINE GENERATES RANDOM DATA
>150 D=RND
>160 RETURN
>170 REM THIS SUBROUTINE SUMS THE DATA
>180 SU=SU+D
>190 RETURN
>200 REM THIS SUBROUTINE AVERAGES THE DATA
>210 AV=SU/I
>220 RETURN
READY
>BRKPNT 160
Breakpoint enabled.
READY
>RUN
STOP - IN LINE 160
READY
```

CONT



Use the CONT command to resume execution of a program stopped by a  +  (page 10 -4), BRKPNT(page 10 -2), SINGLSTP (page 10 -20), or a STOP (page 11 -36). If you stop a program by pressing  +  on the console device or by executing a STOP statement, you can resume execution of the program by typing CONT. Between stopping and re-starting the program you may display the values of variables or change the values of variables. However, you cannot continue if you modify the program during the STOP or after an error.





Important:  +  clears all input and output buffers.

Syntax



CONT 

Example



```
>10 FOR I = 1 TO 10000
>20 PRINT I
>30 NEXT I
>40 END
READY
>RUN
1
2
3
4
5
6
7
8
9
10  + 
STOP - IN LINE 30
READY
>CONT
20
21
22
```



Notice that after  +  is pressed and I is printed the value of I is 20. The value of I is incremented several times before  +  is detected.

CTRL-C

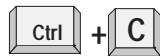
Use the  +  command to stop execution of the current program and return the BASIC module to the Command mode. In some cases you can continue execution of the program using a CONT (page 10 -3).

Important:  +  clears all input and output buffers.



Use CALL 19 (page 12 -12) to disable  + .





Use CALL 18 (page 12 -11) to re-enable  + .

Syntax

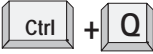
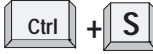


Example

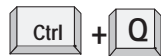
```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 10000
>20 PRINT I
>30 NEXT I
>40 END
>RUN
  1
  2
  3
  4
  5 -  + 
STOP - IN LINE 20
READY
>PRINT I
  27
>I = 10
>CONT
  10
  11
  12
```

Notice that after  +  is pressed and I is printed the value of I is 27. The value of I is incremented several times before  +  is detected.

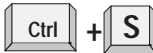
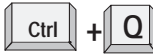
CTRL-Q

If software handshaking is enabled on the program port (page 2 -4), use the  command to restart a LIST command (page 10 -9) or PRINT output (page 11 -29) that is interrupted by  (page 10 -6) .

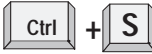
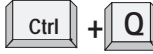
Syntax









Example



```
> LIST
1  REM EXAMPLE PROGRAM
10 A = 1
20 DO

.
.
.

30 A = A+1
40 PRINT A
50 WHILE A < 20



READY
>
```

In this example, the output is suspended when  is pressed. The output is continued after  is pressed.



CTRL-S

If software handshaking is enabled on the program port (page 2 -4), use the  +  command to interrupt the scrolling of a BASIC program during the execution of a LIST command.  +  stops output from the transmitting port if you are running a program. In this case XOFF  +  operates as follows:





- XOFF only operates on PRINT statements (page 11 -29).
- When received during a PRINT, data output is suspended immediately, but program execution continues until the buffer is full and then the program is suspended.
- When received at any other time, the program continues until encountering a PRINT statement. At this time data output is suspended. The program continues to fill the output buffer until the buffer is full and then the program is suspended.
- XON  +  (page 10 -5) resumes data output operation.





Important:  +  only works if you have enabled software handshaking on the program port. You enable software handshaking through the MODE statement (page 11 -20). Software handshaking is enabled by default.

Syntax

 + 
















Example

```
>10 A = 1
>20 DO
 + 
.
.
.
 + 
>30 A = A+1
>40 PRINT A
>50 WHILE A < 20
READY
```


In this example, the output is suspended when  +  is pressed. The output is continued after  +  is pressed.

EDIT

Use the EDIT command to access the BASIC line editor. Use this editor to edit a line of the current program in RAM.

Operation	Function	Key Strokes
delete	use the delete operation to delete the character at the cursor position	 + 
exit	use the exit operation(s) to exit the editor with or without saving the changes	 +  - exits the editor and replaces the old line with the edited line  +  - exits the editor without saving any changes made to the line
insert	use the insert operation to insert text at the current cursor position Important: When you use the insert operation, all text to the right of the cursor disappears until you press the second  +  total line length is 79 characters.	 +  Important: You must press a second  +  to terminate the insert command
move	use the move operation to provide right/left cursor control	 - moves the cursor one space to the right  - moves the cursor one space to the left
replace	use the replace operation to replace the character at the current cursor position	press the key that corresponds to the character that replaces the character at the current cursor position
retype	use the retype operation to copy the current line of text and insert it at the line following the current line. The cursor moves to the first character on the new line	

Syntax

`EDIT line number` 

Example

`>EDIT 150`

Result: Displays program line number 150 for editing.

ERASE

Use the ERASE command to delete the last BASIC program stored in EEPROM through a PROG command (page 10 -11).

Syntax

```
ERASE 
```

Example

```
>ERASE  
>ERASED ROM 13
```

Result: The last program stored in EEPROM (ROM 13 in this example) is erased.

LIST

Use the LIST command to print the program to the console device. Spaces are inserted after the line number and before and after statements. This helps in the debugging of BASIC module programs. You can terminate the listing of a program at any time by pressing **Ctrl** + **C** (page 10 -4) on the console device. You can interrupt and continue the listing using **Ctrl** + **S** (page 10 -6) and **Ctrl** + **Q** (page 10 -5).

Important: **Ctrl** + **C** terminates the listing if **Ctrl** + **C** checking is enabled. **Ctrl** + **S** halts the listing until **Ctrl** + **Q** is pressed if software handshaking is enabled.

Syntax

Command variation	Description
LIST Return	lists the entire program
LIST <i>ln num</i> Return	lists the line number specified to the end of the program
LIST <i>ln num - ln num</i> Return	lists the program from the first designated line number to the second designated line number Important: you must separate the two line numbers with a dash (-)

Use the **LIST@** command to print the program to the device attached to port PRT1. Use the **LIST#** command to print the program to the device attached to port PRT2. You must configure the port parameters to match your particular list device. These parameters (communication rate, parity, stop bits, and so on) can be set using the MODE statement (page 11 -20).

Example

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

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

>LIST 30
30 PRINT I
40 NEXT I
50 END
READY
```

NEW

Use the NEW command to delete the program and all variables currently stored in RAM. In addition, all variables are set equal to ZERO; all strings and all BASIC evoked interrupts are cleared. The free running clock, string allocation, and internal stack pointer values are not affected.

Syntax

```
NEW 
```


Example

```
>NEW  
>LIST  
READY  
>
```

NULL

Use the NULL command to set how many null characters (00H) you want the BASIC module to output after a carriage return in a print statement (page 11 -29). After initialization this value is set to 0. Most printers contain a RAM buffer that eliminates the need to output null characters after a carriage return.

Syntax

```
NULL (integer) 
```

Example

```
>NULL (10)
```

PROG

Important: Before you attempt to program an EEPROM, read the PROG, PROG1, PROG2, and MODE sections of this chapter. See also CALL 78 (page 13 -8)

Use the PROG command to program the resident EEPROM with the current program in RAM. BASIC module cannot program EPROMs; however, it can read them. After you type PROG, the BASIC module displays the number the program occupies in the EEPROM FILE. Programming only takes a few seconds.

Important: Be sure you have selected the program you want to save before using the PROG command. Your BASIC module does not automatically copy the RAM program to EEPROM. If an error occurs during EEPROM programming, the message ERROR: Programming sequence failure is displayed.

Important: If you exceed the available EEPROM space, you cannot continue programming until it is erased. Use the ERASE command (page 10 -8) to erase the last program stored in EEPROM. Be sure to use CALL 81 (page 13 -10) or CALL 82 (page 13 -11) to determine memory space prior to programming EEPROM.

Syntax

PROG 

Example

```
>LIST
1  REM EXAMPLE PROGRAM
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
40 END
READY
>PROG
ROM 12
PROGRAMMING SEQUENCE WAS SUCCESSFUL
READY
>ROM 12
>LIST
1  REM EXAMPLE PROGRAM
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
40 END
READY
```

Result: The program placed in the EEPROM is the 12th program stored.

PROG1

Important: Before you attempt to program an EEPROM, read the PROG, PROG1, PROG2, and MODE sections of this chapter. Also see CALL 78 (page 13 -8). If you have already used a PROG2, PROG1 supercedes it.

Use the PROG1 command to program the resident EEPROM with port information for all three ports as well as store MTOP (page 9 -18) information. At module power up, the BASIC module reads this information and initializes MTOP and all three serial ports. The sign-on message is sent to the console immediately after the module completes its reset sequence. If the communication rate on the console device changes, you must re-program the EEPROM to make the module compatible with the new console. Re-program by changing the appropriate port or MTOP information, then execute PROG1 again.

The flowchart on page 10 -14 shows the BASIC module operation from a power-up condition using PROG1 and PROG2, or battery backed RAM.

Syntax

PROG1 

Example

```
READY  
>PROG1  
PROGRAMMING SEQUENCE WAS SUCCESSFUL
```

PROG2

Important: Before you attempt to program an EEPROM, read the PROG, PROG1, PROG2, and MODE sections of this chapter. Also see CALL 78 (page 13 -8). If you have already used a PROG1, PROG2 supercedes it.

Note, the PROG2 command does not transfer the RAM program to EEPROM. The PROG2 command enables the first program in EEPROM to be run at each power up. The PROG2 command is the same as the PROG1 command except the module immediately begins executing the first program stored in the resident EEPROM instead of signing on and entering the Command mode.

Use the PROG2 command to program the resident EEPROM with port information for all three ports as well as store MTOP (page 9 -18) information. At module power up, the BASIC module reads this information and initializes MTOP and all three serial ports. The module immediately begins executing the program stored in RAM. Otherwise, if there is not a program in RAM, it executes the first program stored in the EEPROM. Re-program by changing the appropriate port or MTOP information, then execute PROG2 again.

You can use the PROG2 command to RUN (page 10 -19) a program on power up without connecting to a console. Once you use PROG2 (and JW4 is not in the default position (page 1 -6)), the module powers up and executes a RUN command if a program is in RAM. If there is not a program in RAM, but there is a program in ROM, the module powers up and executes a RROM command. If there is not a program in RAM or ROM, the module powers up in the Command mode. This feature also allows you to write a special initialization sequence in BASIC and generate a custom sign-on message for specific applications. PROG2 does not alter the first program in the memory module.

Important: The PROG2 command does not cause the BASIC module to RUN at power up if PRT1 default communications are selected via JW4 (page 1 -6).

Important: PROG2 only executes ROM1 if no program is in RAM.

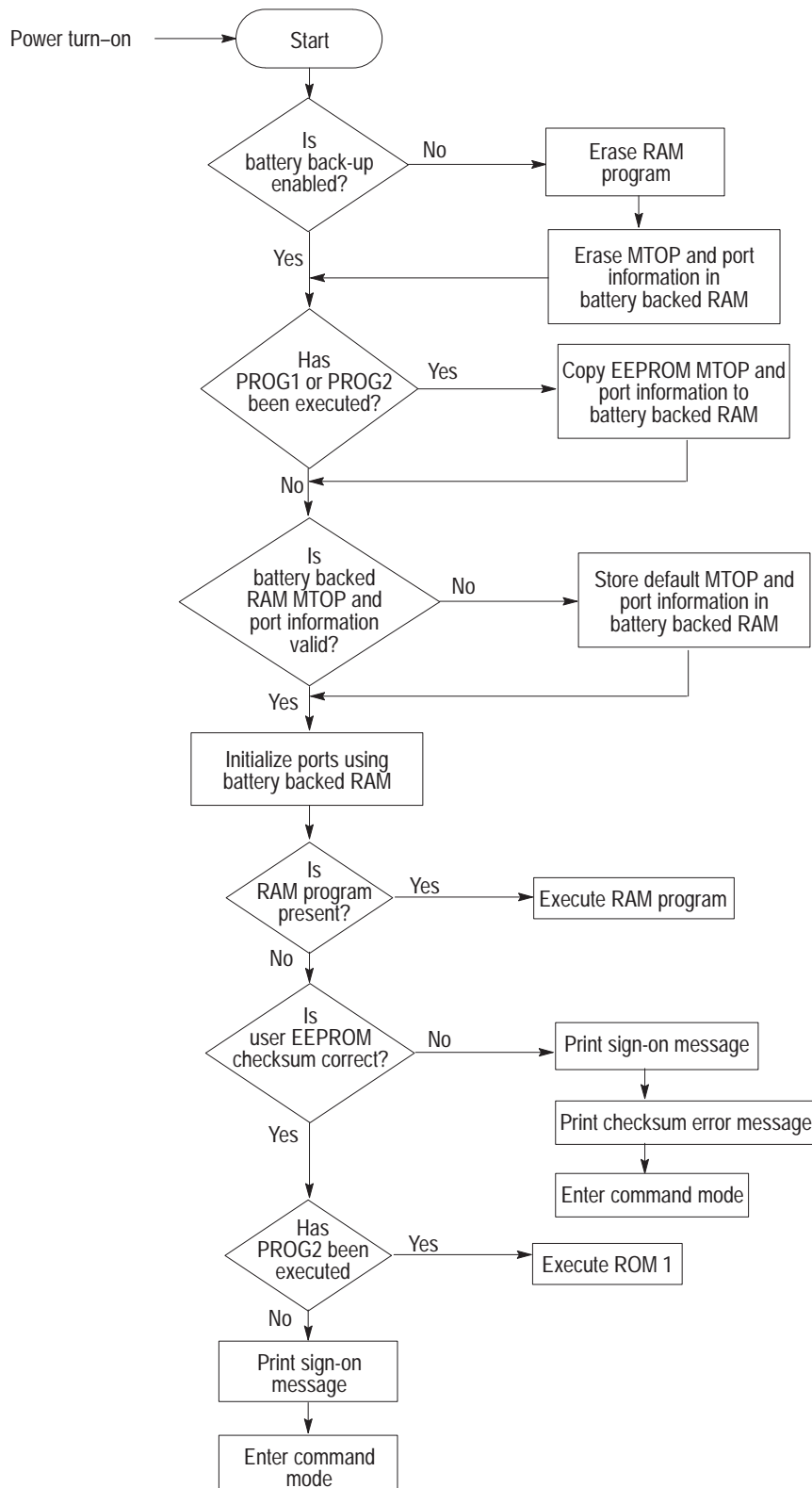
The flowchart on page 10 -14 shows the BASIC module operation from a power-up condition using PROG1 and PROG2, or battery backed RAM.

Syntax

```
PROG2 
```

Example

```
READY
>PROG2
PROGRAMMING SEQUENCE WAS SUCCESSFUL
```



RAM

Tip

Use the RAM command to tell the BASIC module interpreter to select the current program out of RAM. Use a LIST command (page 10 -9) to display and a RUN command (page 10 -19) execute the current program.

The execution time for a program running in RAM is the same as a program running from ROM. There is no performance improvement on a BASIC program by moving it to RAM.

Important: RAM space is limited to 24K bytes. Use this formula to calculate the available user RAM space:

$$\text{Available user RAM} = \text{MTOP (see page 9 -18)} - H$$

$$H = \text{LEN} + S + 6 * A + 8 * V + 512$$

Where:

LEN	system control value that contains current RAM program length (see also page 9 -17)
S	number of bytes allocated for strings (first value in the STRING instruction)
A	sum of all (array sizes + 1)
V	sum of all variable names used (including each array name)

Syntax

RAM 

Example

```
READY
>RAM
```

REN





Use the REN command to renumber program lines.



Important:

- REN command updates the destination of GOSUB, GOTO, ONERR, ONTIME and ON GOTO statements (Chapter 11).
- If the target line number does not exist, or if there is insufficient memory to complete the task, no lines are changed and the message `RENUMBER ERROR` appears on the console screen.
- Because the REN command uses the same RAM for renumbering as it does for variable and program storage, available RAM may be insufficient in large programs. Renumber your program periodically and in segments during development.

Syntax

This command variation:	Renumbers the program:
<code>REN</code> 	from the beginning of the program. New line numbers begin at 10 and increment by 10.
<code>REN NUM</code> 	from the beginning of the program. New line numbers begin at 10 and increment by NUM.
<code>REN NUM1, NUM2</code> 	from the beginning of the program. New line numbers begin with NUM1 and increment by NUM2.
<code>REN NUM1, NUM2, NUM3</code> 	starting at NUM2. New line numbers begin with NUM1 and increment by NUM3.

Examples

`>REN`

Result: Renumbers the entire program. First new line number is 10. Line numbers increment by 10.

`>REN 20`

Result: Renumbers the entire program. The first new line number is 10. Line numbers increment by 20.

`>REN 300,50`

Result: Renumbers the entire program. The first new line number is 300. Line numbers increment by 50.

`>REN 1000,900,20`

Result: Renumbers the program from line 900 on up. Line number 900 becomes line number 1000. Any following line numbers increment by 20.

ROM

Use the ROM command to tell the BASIC module interpreter to select the current program out of EEPROM or EPROM. Use a LIST command (page 10 -9) to display and a RUN command (page 10 -19) to execute the current program.

Tip

The execution time for a program running in ROM is the same as a program running from RAM. There is no performance improvement on a BASIC program by moving it to RAM.

Important: Your BASIC module can execute and store up to 255 programs in EEPROM depending on the size of the programs and the capacity of the EEPROM. The programs are stored in a sequence string, referred to as the EEPROM file, in EEPROM for retrieval and execution.

When you enter ROM *integer*, the BASIC module selects that program out of EEPROM memory and makes it the current program. If no integer is typed after the ROM command (ex. ROM) the module defaults to ROM 1. Since the programs are stored in sequence in EEPROM, the integer following the ROM command selects the program you want to run or list. If you attempt to select a program that does not exist (ex. you type ROM 8 and only 6 programs are stored in the EEPROM) the message ERROR: PROM MODE is displayed.

The module does not transfer the program from EEPROM to RAM when you select the ROM mode. If you attempt to alter a program in the ROM mode by typing in a line number, the message ERROR: PROM MODE is displayed. The XFER command (page 10 -21) allows you to transfer a program from EEPROM to RAM for editing purposes.

Important: When you transfer programs from EEPROM to RAM you lose the previous RAM contents.

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 move back and forth between the two modes when in Command mode. If you are in Run mode, you can change back and forth using CALLs 70 (page 13 -2), 71 (page 13 -3), and 72 (page 13 -4). You can also use all of the RAM for variable storage if the program is stored in EEPROM. The system control value MTOP (page 9 -18) always refers to RAM. The system control value LEN (page 9 -17) refers to the program currently in RAM or ROM.

Syntax

ROM *integer* 

Example

```
READY
>ROM 1
```

RROM

Use the RROM command to tell the BASIC module interpreter to select the current program out of EEPROM or EPROM and then execute the program. This command is equivalent to typing ROM (page 10 -17) and then RUN (page 10 -19).

Tip

The execution time for a program running in ROM is the same as a program running from RAM. There is no performance improvement on a BASIC program by moving it to RAM.

Important: Your BASIC module can execute and store up to 255 programs in EEPROM depending on the size of the programs and the capacity of the EEPROM. The programs are stored in a sequence string, referred to as the EEPROM file, in EEPROM for retrieval and execution.


When you enter RROM *integer*, the BASIC module selects that program out of EEPROM memory, makes it the current program, and starts program execution. If no integer is typed after the RROM command (example: RROM) the module defaults to RROM 1. Since the programs are stored in sequence in EEPROM, the integer following the RROM command selects the program you want to run or list. If you attempt to select a program that does not exist (ex. you type RROM 8 and only 6 programs are stored in the EEPROM) the message ERROR: PROM MODE is displayed.

The module does not transfer the program from EEPROM to RAM when ROM mode is selected. If you attempt to alter a program in ROM mode by typing in a line number, the message ERROR: PROM MODE is displayed. The XFER command (page 10 -21) allows you to transfer a program from EEPROM to RAM for editing purposes.

Important: When you transfer programs from EEPROM to RAM you lose the previous RAM contents.

Since the RROM command does *not* transfer a program to RAM, it is possible to have different programs in ROM and RAM simultaneously. You can move back and forth between the two modes when in Command mode. If you are in Run mode, you can change back and forth using CALLs 70 (page 13 -2), 71 (page 13 -3), and 72 (page 13 -4). You can also use all of the RAM for variable storage if the program is stored in EEPROM. The system control value MTOP (page 9 -18) always refers to RAM. The system control value LEN (page 9 -17) refers to the program currently in RAM or ROM.





Syntax

```
RROM integer 
```

Example

```
READY
>RROM 2
```

RUN

Use the RUN command to set all variables equal to zero, clear all BASIC evoked interrupts, and begin program execution with the first line number of the selected program. The RUN, CONT (page 10 -3), and RROM (page 10 -18) commands and the GOTO statement (page 11 -14) are the only ways you can place the BASIC module interpreter into Run mode from Command mode. Terminate program execution at any time by pressing  +  (page 10 -4) on the console device. As long as you have not used CALL 19 (page 12 -12) to disable  + .

Important: Some BASIC interpreters allow a line number to follow the RUN command (example: RUN 100). The BASIC module does not permit this variation on the RUN command. Execution begins with the first line number. To obtain a function similar to the RUN *ln num* command, use the GOTO *ln num* statement in the Command mode.

Syntax

RUN 


Example

```
>1 REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 PRINT I
>30 NEXT I
>40 END
>RUN
  1
  2
  3
READY
>
```

SNGLSTP

Use the SNGLSTP command to initiate single-step program execution. If the number you specify with this command is zero, single-step execution is disabled. If the number is not zero, a break point is set before each line in the program. Start the program with the RUN command (page 10 -19). After each stop, type CONT (page 10 -3) to execute the next line. You can inspect variables or assign variables at each break point. SNGLSTP works only on programs executing from RAM. It does not stop a program executing from ROM.

Syntax

```
SNGLSTP integer 
```

Example

```
>10 FOR I = 1 TO 5
>20 PRINT I
>30 NEXT I
>40 PRINT "PASSED FOR - NEXT LOOP"
>50 PRINT "THIS IS THE END"
>60 END
READY
>SNGLSTP 20
SINGLE STEP ENABLED
READY
>RUN
STOP - LINE 20
READY
>CONT
1
STOP - LINE 30
READY
>CONT
STOP - LINE 20
READY
>CONT
2
STOP - LINE 30
READY
>SNGLSTP 0
SINGLE STEP DISABLED
READY
>CONT
3
4
5
PASSED FOR - NEXT LOOP
THIS IS THE END
READY
```

VER

Use the VER command to print the BASIC module sign-on message that displays the current version of the firmware.

Syntax

VER 

Example

```
READY  
>VER  
PLC BASIC Module-Catalog Number 1771-DB/B  
Firmware Revision: A  
Allen-Bradley Company, Copyright 1989, 1990, 1991, 1992, 1993 1994  
All rights reserved
```

XFER

Use the XFER command to transfer the current selected program in ROM to RAM and select RAM mode. After the XFER command executes, you can edit the program in the same way you edit any RAM program.

Important: The XFER command clears existing RAM programs.

Syntax

XFER 

Example

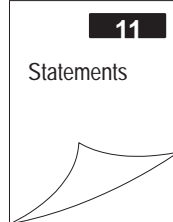
```
READY  
>XFER
```

Command Line Calls

These calls can only be executed from the command line. Use these calls to cause a function to occur within the BASIC module.

CALL	Description	Page
31	display PRT2 port parameters	12 -21
73	battery-backed RAM disable	13 -5
74	battery-backed RAM enable	13 -5
77	protected variable storage	13 -6
81	user PROM check and description	13 -10
82	check user memory module map	13 -11
83	display DH485 port setup	13 -11
94	display current PRT1 port setup	13 -32
101	upload user (E)EPROM code to host	13 -35
103	print PRT1 transmit buffer and pointer	13 -36
104	print PRT1 receive buffer and pointer	13 -37
109	print the argument stack	13 -44
110	print the PRT2 transmit buffer and pointer	13 -45
111	print the PRT2 receive buffer and pointer	13 -46

What's Next?



Statements

What's in This Chapter?



BASIC statements are programming instructions that control program flow or manipulate I/O or memory. Every statement begins with a line number, followed by a statement body, and terminated with a carriage return (CR) or a colon (:) in case of multiple statements per line number. You execute statements automatically within a BASIC program during Run mode. You can also enter these statements from the command line in Command mode to test/evaluate the execution of the statement. Chapter 7 gives you an overview of how to use the different statements within your BASIC program.

This chapter describes:	On page:	This chapter describes:	On page:
CALL	Ch. 12 & 13	LET	11 -19
CLEAR	11 -2	MODE	11 -20
CLEARI	11 -3	NEXT	11 -21
CLEARs	11 -3	ONDF1	11 -22
CLOCK0	11 -4	ONERR	11 -23
CLOCK1	11 -5	ON-GOSUB	11 -24
DATA	11 -6	ON-GOTO	11 -26
DIM	11 -7	ONTIME	11 -25
DO-UNTIL	11 -8	PH0. and PH1.	11 -27
DO-WHILE	11 -9	POP	11 -28
END	11 -10	PRINT	11 -29
FOR-TO-(STEP)-NEXT	11 -11	PUSH	11 -30
GET	11 -12	READ	11 -31
GOSUB	11 -13	REM	11 -32
GOTO	11 -14	RESTORE	11 -32
IDLE	11 -14	RETI	11 -33
IF-THEN-ELSE	11 -15	RETURN	11 -34
INPL	11 -16	ST@	11 -35
INPS	11 -16	STOP	11 -36
INPUT	11 -17	STRING	11 -37
LD@	11 -18	what's next?	11 -38

CLEAR

Use the CLEAR statement to set all variables equal to 0 and reset all BASIC evoked interrupts and stacks. This means that after you execute the CLEAR statement, you must execute an ONTIME statement (page 11 -25) before the module acknowledges the internal timer interrupts. ERROR trapping with the ONERR statement (page 11 -23) does not re-occur until you execute an ONERR *In num* statement.

The CLEAR statement does not affect the free running clock that is enabled by the CLOCK1 statement (page 11 -5). CLOCK0 (page 11 -4) is the only module statement that can disable the free running clock. Nor does CLEAR reset the memory that has been allocated for strings, so it is not necessary to re-enter the STRING *expr; expr* statement (page 11 -37) to re-allocate memory for strings after executing the CLEAR statement. In general, use CLEAR to erase all variables.

You can also use CLEAR in Command mode.

Syntax

```
CLEAR
```

Example

```
>CLEAR
>LIST
1  REM EXAMPLE PROGRAM
10 DIM A(4)
20 DATA 10,20,30,40
30 FOR I=0 TO 3
40 READ A(I)
50 NEXT I
60 FOR J=0 TO 3
70 PRINT A(J)
80 NEXT J
READY
>PRINT A(1),I,J
0 0 0
>RUN
10
20
30
40
READY
>PRINT A(1),I,J
20 4 4
>CLEAR
>PRINT A(1),I,J
0 0 0
```

CLEARI

Use the CLEARI statement to clear all of the BASIC evoked interrupts. The ONTIME (page 11 -25) interrupt is disabled after you execute the CLEARI statement.

The CLEARI statement does not affect the free running clock that is enabled by the CLOCK1 statement (page 11 -5). CLOCK0 (page 11 -4) is the only module statement that can disable the free running clock.

You can use this statement to selectively disable ONTIME interrupts during specific sections of your BASIC program. You must execute the ONTIME statement again before the specific interrupts are enabled.

You can also use CLEARI in Command mode.

Important: This statement does not clear CALL 32, CALL 16 or ONDF1.

Syntax

```
CLEARI
```

Example

```
READY  
>CLEARI
```

CLEARs

Use the CLEARs statement to reset all of the stacks. The control, argument, and internal stacks all reset to their initialization values. You can use this command to reset the stacks if an error occurs in a subroutine.

You can also use CLEARs in Command mode or if you wish to exit a loop (i.e., GOSUB, DO WHILE, FOR NEXT, etc.).

Syntax

```
CLEARs
```

Example

```
READY  
>CLEARs
```

CLOCK0

Use the `CLOCK0` (zero) statement to disable or turn off the free running clock resident on the `BASIC` module. After you execute `CLOCK0`, the special function operator `TIME` (page 9 -19) no longer increments. `CLOCK0` is the only module statement that can disable the free running clock. `CLEAR` (page 11 -2) and `CLEARI` (page 11 -3) do *not* disable the free running clock, only its associated `ONTIME` interrupt (page 11 -25).

Important: `CLOCK0` and `CLOCK1` (page 11 -5) are independent of the clock/calendar.

You can also use `CLOCK0` in Command mode.

Syntax

```
CLOCK0
```

Example

```
READY  
>CLOCK0
```

CLOCK1

Use the CLOCK1 statement to enable the free running clock resident on the BASIC module. The special function operator TIME (page 9 -19) increments once every 5 milliseconds after you execute CLOCK1. CLOCK1 uses an internal timer to generate an interrupt once every 5 milliseconds. Because of this, the special function operator TIME has a resolution of 5 milliseconds. 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 0. The interrupts associated with CLOCK1 cause the module programs to run at about 99.6% of normal speed. This means that the interrupt handling for the free running clock uses about 0.4% of the total CPU time.

Important: This does not include additional overhead for ONTIME user interrupt handling execution.

Syntax

```
CLOCK1
```

Example

```
>10 TIME = 0
>15 DBY(71) = 0 :REM RESET NONINTEGER PORTION OF TIME
>20 CLOCK1
>30 ONTIME 2,100
>40 DO
>50 WHILE TIME < 10
>60 END
>100 PRINT "TIMER INTERRUPT AT - ",TIME," SECONDS"
>110 ONTIME TIME+2,100
>120 RETI
READY
>RUN

TIMER INTERRUPT AT - 2.01 SECONDS
TIMER INTERRUPT AT - 4.015 SECONDS
TIMER INTERRUPT AT - 6.01 SECONDS
TIMER INTERRUPT AT - 8.01 SECONDS
TIMER INTERRUPT AT - 10.01 SECONDS
```

DATA

Use the DATA statement to specify the expressions that you can retrieve with a READ statement (page 11 -31). If you use multiple expressions per line, you *must* separate them with a comma.

Every time a READ statement is encountered the next consecutive expression in the DATA statement is evaluated and assigned to the variable in the READ statement. You can place DATA statements anywhere within a program. They are not executed and do not cause an error.

DATA statements are chained and appear as one large DATA statement. If at anytime all the data is read and you execute another READ statement, the program terminates and the message ERROR: NO DATA - IN LINE XX prints to the console device. The module returns to Command mode.

Important: You cannot use the CHR operator (page 9 -16) in a DATA statement.

See also RESTORE (page 11 -32)

Syntax

DATA

Example

```
>LIST
1  REM EXAMPLE PROGRAM
10 DIM A(4)
20 DATA 10,ASC(A),ASC(C),35.627
30 FOR I=0 TO 3
40 READ A(I)
50 NEXT I
60 FOR J=0 TO 3
70 PRINT A(J)
80 NEXT J

READY
>RUN
10
65
67
35.627
```

DIM

Use the DIM statement to reserve storage for arrays. The storage area is first assumed to be zero. Arrays in the BASIC module may have only one dimension and the size of the dimensioned array may not exceed 254 elements.

Once you dimension a variable in a program you may not re-dimension it. An attempt to re-dimension an array causes an array size error and the module enters the Command mode.

If you use an array variable that you did not dimension by a DIM statement, BASIC assigns a default value of 10 to the array size. All arrays are set equal to zero when you execute the RUN command (page 10 -19), NEW command (page 10 -10) or the CLEAR statement (page 11 -2).

The number of bytes allocated for an array is six times the array size plus one ($6 * (\text{array size} + 1)$). For example, the array A (100) requires 606 ($6 * (100+1)$) bytes of storage. Memory size usually limits the size of a dimensioned array.

Important: If using strings you must define them in your program with the STRING statement (page 11 -37) before you define arrays with the DIM statement.

Syntax

DIM

Example

```
>1 REM EXAMPLE PROGRAM
>10 DIM A(25), C(15), A1(20)
```

Error on attempt to re-dimension array:

```
>1 REM EXAMPLE PROGRAM
>10 A(5) = 10 :REM BASIC ASSIGNS DEFAULT OF 10 TO ARRAY A
>20 DIM A(5) : REM ARRAY RE-DIMENSION ERROR
READY
>RUN
ERROR: ARRAY SIZE - IN LINE 20
20 DIM A(5) : REM ARRAY RE-DIMENSION ERROR
-----X
READY
>
```

DO-UNTIL

Use the DO-UNTIL statement to set up loop control within a module program. All statements between the DO and the UNTIL *rel expr* are executed until the relational expression following the UNTIL statement is true. You can nest DO-UNTIL loops.

The control stack (C-stack) stores all information associated with loop control. The C-stack is 157 bytes long. DO-UNTIL loops use 3 bytes of the C-stack.

Do not improperly exit this loop or a C-stack error occurs. See CLEAR (page 11 -3).

Important: Excessive nesting exceeds the limits of the C-stack, generating an error, and causing the module to enter Command mode. For more information see control stack, page 8 -1.

Syntax

```
DO-UNTIL rel expr
```

Example

Simple DO-UNTIL

```
>1  REM EXAMPLE PROGRAM
>10 A=0
>20 DO
>30 A=A+1
>40 PRINT A
>50 UNTIL A=4
>60 PRINT "DONE"
>70 END
>RUN
1
2
3
4
DONE
>READY
```

Nested DO-UNTIL

```
>1  REM EXAMPLE PROGRAM
>10 DO
>20 A=A+1
>30 DO
>40 C=C+1
>50 PRINT A,C,A*C
>60 UNTIL C=3
>70 C=0
>80 UNTIL A=3
>90 END
>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
```


DO-WHILE

Use the DO-WHILE statement to set up loop control within a module program. This statement is similar to the DO-UNTIL *rel expr* (page 11 -8). All statements between the DO and the WHILE *rel expr* are executed as long as the relational expression following the WHILE statement is true. You can nest DO-WHILE statements.

The control stack (C-stack) stores all information associated with loop control. The C-stack is 157 bytes long. DO-WHILE loops use 3 bytes of the C-stack. Do not improperly exit this loop or a C-stack error occurs. See CLEARS (page 11 -3).

Important: Excessive nesting exceeds the limits of the C-stack. An error occurs and the module enters Command mode. For more information see control stack, page 8 -1.

Syntax

```
DO-WHILE rel expr
```

Example

Simple DO-WHILE

```
>10 DO
>20 A = A + 1
>30 PRINT A
>40 WHILE A < 2
>50 END
READY
>RUN
1
2
```

Nested DO-WHILE

```
>10 A=0 : C=0
>20 DO
>30 A=A+1
>40 DO
>45 C=C+1
>50 PRINT A,C,A*C
>60 WHILE C<>3
>70 C=0
>80 WHILE A<4
>90 END
READY
>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
4 1 4
4 2 8
4 3 12
```

END

Use the END statement to terminate program execution. CONT (page 10 -3) does not operate if you use the END statement to terminate execution. An ERROR : CAN'T CONTINUE prints to the console. Always include an END statement to properly terminate a program.

Syntax

```
END
```

Example

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END

READY
>RUN

1 2 3 4
```

FOR-TO-(STEP)-NEXT

Use the FOR-TO-(STEP)-NEXT statement to set up and control program loops.

If the STEP statement and the value are omitted, the increment value defaults to 1, therefore; STEP is an optional statement. The NEXT statement returns the loop to the beginning of the loop and adds the value of the STEP *expr* to the current index value. The current index value is then compared to the limit value of the index.

If the index is less than or equal to the limit, control transfers back to the statement after the FOR statement. Stepping backward (FOR I = 100 TO 1 STEP -1) is permitted in the BASIC module. The NEXT statement is always followed by the appropriate variable.

The control stack (C-stack) stores all information associated with loop control. The C-stack is 157 bytes long. FOR-NEXT loops use 17 bytes of the C-stack. Do not improperly exit this loop or a C-stack error occurs. See CLEAR\$ (page 11 -3).

Important: Excessive nesting exceeds the limits of the C-stack, generating an error, and causing the module to enter Command mode.

For more information see control stack, page 8 -1.

Syntax

```
FOR expr TO expr STEP expr
.
.
NEXT expr
```

Example

```
>5 E=0 : C=10 : D=2
>10 FOR A=E TO C STEP D
>20 PRINT A
>30 NEXT A
>40 END
>RUN
0
2
4
6
8
10
```

Since E=0, C=10, D=2, and the PRINT statement at line 20 executes 6 times, the values of A that are printed are 0, 2, 4, 6, 8 and 10. A represents the name of the index or loop counter. The value of E 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 to the index.

GET

Use the GET statement in the Run mode. GET returns a result of zero in the Command mode. The GET operator reads the console input device. If a character is available from the console device, the value of the character is assigned to GET. After GET is read in the program, it is assigned the value of zero until another character is sent from the console device.

The GET statement is read only once before it is assigned a value of zero. This guarantees that the first character entered is always read, independent of where you place the GET statement in the program. The program port is buffered with a 256 byte circular transmit queue and 256 byte circular receive queue.

Use the GET# operator to read port PRT2 and the GET@ operator to read port PRT1.

Syntax

```
GET  
GET#  
GET@
```

Example

```
>10 A = GET  
>20 IF (A<>0) THEN PRINT A : REM ZERO MEANS NO ENTRY  
>30 GOTO 10  
>RUN  
  
65 [A]  
49 [1]  
24 [^X]  
50 [2]  
  
STOP - IN LINE 30  
READY  
>
```

GOSUB

Use the GOSUB statement to cause the BASIC module to transfer control of the program to the line number the GOSUB statement references. In addition, the GOSUB statement saves the location of the next statement after the GOSUB on the C-stack, so that you can perform a RETURN statement (page 11 -34) to return control to that statement after the GOSUB executes. You may nest the GOSUB statement.

The control stack (C-stack) stores all information associated with loop control. The C-stack is 157 bytes long. GOSUB statements use 3 bytes of the C-stack. Do not improperly exit this subroutine or a C-stack error may occur. See CLEARS (page 11 -3).

Important: Excessive nesting exceeds the limits of the C-stack, generating an error, and causing the module to enter Command mode. For more information see control stack, page 8 -1.

Important: Always use a RETURN (page 11 -34) to exit a GOSUB. C-stack errors may occur if you do not use a RETURN.

Syntax

```
GOSUB ln num
```

Example

Simple Subroutine

```
>10 FOR I = 1 TO 3
>20 GOSUB 100
>30 NEXT I
>40 END
>100 PRINT I
>110 RETURN
READY
>RUN
1
2
3
```

Nested Subroutine

```
>10 FOR I = 1 TO 3
>20 GOSUB 100
>30 NEXT I
>40 END
>100 REM USER SUBROUTINE HERE
>105 PRINT I,
>110 GOSUB 200
>120 RETURN
>200 REM START OF NESTED SUBROUTINE
>210 PRINT I*I
>220 RETURN
READY
>RUN
1 1
2 4
3 9
```

GOTO

Use the GOTO statement to cause BASIC to transfer control to the line number you specify. If line number you specify does not exist, the message `ERROR: INVALID LINE NUMBER` is printed to the console device and the BASIC module enters the Command mode.

Unlike the RUN command (page 10 -19), if you execute the GOTO statement in the Command mode it does not clear the variable storage space or interrupts. However, if you execute the GOTO statement is in the Command mode after you edit a line, the module clears the variable storage space and all BASIC evoked interrupts.

Syntax





```
GOTO ln num
```

Example

```
>1 REM EXAMPLE PROGRAM
>50 GOTO 100
```

If line 100 exists, this statement causes execution of program to resume at line 100.

IDLE

Use the IDLE statement to force the BASIC module to wait until an interrupt occurs (i.e., ONTIME, ONERR, etc.). Program execution halts until an ONTIME condition is met. The ONTIME (page 11 -25) interrupt must be enabled before executing the IDLE command or else the BASIC module enters a wait forever mode.  +  (page 10 -4) exits the IDLE command if  +  is enabled. (See CALL 18, page 12 -11).

Important: Do not use IDLE while executing background calls (page 7 -6). Background calls do not execute properly if IDLE is running.

Syntax

```
IDLE
```

Example

```
>10 TIME = 0
>20 CLOCK1
>30 ONTIME 2,70
>40 IDLE
>50 PRINT "END OF TEST!!!"
>60 END
>70 PRINT "TIMER INTERRUPT AT - ",TIME,"SECONDS."
>80 RETI
READY
>RUN
TIMER INTERRUPT AT - 2.005 SECONDS.
END OF TEST!!!
```

IF-THEN-ELSE

Use the IF-THEN-ELSE statement to set up a conditional test.

If you want to transfer control to different line numbers using the IF statement, you may omit the GOTO statement. These examples give the same results:

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

You can replace the THEN statement with any valid BASIC module statement:

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

You may execute multiple statements following the THEN or ELSE if you use a colon to separate them:

```
>30 IF A<>10 THEN PRINT A : GOTO 150 ELSE 10
```

You may omit the ELSE statement. If you omit the ELSE statement control passes to the next statement.

```
>1  REM EXAMPLE PROGRAM
>20 IF A=10 THEN 40
>30 PRINT A
```

Syntax

```
IF rel expr THEN valid statement ELSE valid statement
```

Example

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

Upon execution of line 10 IF A is equal to 100, THEN A is assigned a value of 0. IF A does not equal 100, A is assigned a value of A+1.

INPL

Use the INPL statement to read an entire line (up to 254 characters) from the program port buffer. The line must be stored in a string variable. The INPL statement reads all characters from the program port until a carriage return or the 254 character limit is reached, whichever comes first. INPL does not echo characters read from the program port.

Use the INPL# statement to read an entire line of characters from the PRT2 port buffer. Use the INPL@ statement to read an entire line of characters from the PRT1 port buffer. Both these statements function like INPL.

Syntax

```
INPL string_variable
```

Example

```
>1  REM EXAMPLE PROGRAM
>10 STRING 270,254 : REM ONE STRING OF < 254 BYTES
>20 INPL $(0) : REM READ LINE FROM PROGRAM PORT
>30 PRINT# $(0) : REM ECHO STRING TO PORT PRT2
```

INPS

Use the INPS statement to read an entire string of characters from the program port buffer. The INPS statement terminates when the number of characters you specify is reached. The INPS reads CR, commas, and nulls. No characters are echoed. The INPS statement is preferred over INPUT (page 11 -17) or INPL for communications because all ASCII characters may be significant. INPUT is least desirable because input stops when a comma or a carriage return is seen. INPL terminates when a carriage return is seen.

Use the INPS# statement to read an entire string of characters from the PRT2 port buffer. Use the INPS@ statement to read an entire string of characters from the PRT1 port buffer. Both these statements function like the INPS statement.

Syntax

```
INPS string_variable, number_of_characters
```

Example

```
>1  REM EXAMPLE PROGRAM
>100 PRINT, "TYPE P TO PROCEED OR S TO STOP"
>110 REM READ SINGLE CHARACTER FROM PROGRAM PORT
>120 INPS $(0),1
>130 IF ASC$(0),1)= ASC(P) GOTO 500
>140 IF ASC$(0),1)= ASC(S) GOTO 700
>150 GOTO 100
```


INPUT

Use the INPUT statement to enter data from the console device during program execution. You may assign data to one or more variables with a single input statement. You must separate the variables with commas. You are prompted to enter data for each variable after the INPUT. If you do not enter enough data, the module prints TRY AGAIN on the console device.

```
>10 INPUT A,C
```

You can write the INPUT statement so that a descriptive prompt tells you what to enter. The message 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 is not displayed.

```
>10 INPUT "ENTER A NUMBER - ",A
```

You can also assign strings with an INPUT statement. Strings are always terminated with a carriage return (cr). If more than one string input is requested with a single INPUT statement, the module prompts you with a question mark. You can assign strings and variables with a single INPUT statement.

```
>20 INPUT "NAME(CR),AGE - ",$(1),A
```

Use the INPUT# statement to input data from port PRT2. Use the INPUT@ statement to input data from port PRT1. Both these statements function like the INPUT statement.

Syntax

```
INPUT
```

Example

```
>1  REM EXAMPLE PROGRAM
>10 INPUT A,C
>20 PRINT A,C
>RUN
?1
TRY AGAIN
?1,2
1      2
```

LD@

Use the LD@ statement to retrieve floating point numbers stored with a ST@ statement (page 11 -35). The expression following the LD@ statement specifies the address where you want to store the number after executing the LD@. The LD@ statement places the number on the argument stack at the address location you specify with *expr*.

You can use this statement with CALL 77 (page 13 -6) to retrieve variables from a protected area of memory. This protected area is not zeroed on power up or when you issue the RUN command (page 10 -19).

Important: LD@ and ST@ are not used with any port designation.

Syntax

```
LD@ expr
```

Example

```
>PRINT MTOP
24515
>PRINT MTOP 10*6
24455
>PUSH 24455 : CALL 77
>1   REM EXAMPLE PROGRAM
>5   DIM A(10),B(10)
>10  REM *** ARRAY SAVE ***
>20  FOR I = 0 TO 3
>30  A(I) = I+20
>40  PUSH A(I) : REM PUT NUMBER ON STACK
>50  ST@ 5FFFH-I*6
>60  NEXT I
>70  REM *** GET ARRAY ***
>80  FOR I = 0 TO 3
>90  LD@ 5FFFH-I*6
>100 POP B(I) : REM GET NUMBER FROM STACK
>110 PRINT B(I)
>120 NEXT I0
READY
>RUN
20
21
22
23
READY
>PUSH 5FFFH : CALL 77
>PTINT MTOP
24575
```

LET

Use the LET statement to assign a variable to the value of an expression.

The = sign used in the LET statement is not an equality operator. It is a replacement operator. The statement should be read *var* is replaced *expr*. The word LET is always optional (ex. LET A = 2 is the same as A = 2). When LET is omitted the LET statement is called an IMPLIED LET. We use the word LET to refer to both the LET statement and the IMPLIED LET statement.

Also use the LET statement to assign string variables:

```
LET $(1)="THIS IS A STRING"
```

or

```
LET $(2)=$(1)
```

Before you can assign strings you must execute the STRING *expr*; *expr* statement (page 11 -37) or else a memory allocation error occurs that causes the module to enter the Command mode.

Syntax

```
LET var = expr
```

Example

```
>1  REM EXAMPLE PROGRAM  
>10 LET A = 10*SIN(C)/100  
>20 LET B = B +1
```

MODE

Use the MODE command to set the port parameters of ports PRT1, PRT2, and DH-485.

Important: If a argument (other than port name and communication rate) is blank, the argument defaults to the previously specified value for the argument.

PRT1 and PRT2 port parameters (CALL 94, page 13 -32 and CALL 31, page 12 -21 display these settings):

Port parameters	Selections	Default settings
communication rate	300, 600, 1200, 2400, 4800, 9600, 19200	1200
parity (arg1)	none (N), even (E), odd (O)	N
number of data bits (arg2)	7 or 8	8
number of stop bits (arg3)	1 or 2	1
handshaking (arg4)	no handshaking (N) software handshaking (S) hardware handshaking (H) hardware and software handshaking (B)	S
storage type (arg5)	store information in user ROM and RAM (E) ^① store information in battery backed RAM (R)	R

^①The E storage setting type option cannot be used if MODE is used as a statement.

DH-485 port parameters (CALL 83, page 13 -11 displays these settings):

Port parameters	Selections	Default settings
communication rate	300, 600, 1200, 2400, 4800, 9600, 19200	19200
host node address (arg1)	0 to 31	0
module node address (arg2)	1 to 31	1
maximum node address (arg3)	1 to 31	31
not used (arg4)	leave blank	-
storage type (arg5)	store information in user ROM and RAM (E) store information in battery backed RAM (R)	R

Syntax

MODE (*port, communication rate, arg1, arg2, arg3, arg4, arg5*)

Example

```
>1  REM EXAMPLE PROGRAM
>10 MODE(DH485,19200,0,1,2,,R)
.
.
.
>25 MODE(PRT1,1200,N,8,,)
```

NEXT

Use the NEXT statement to return the FOR-TO-(STEP)-NEXT loop (page 11 -11) to the beginning of the loop and add the value of the index increment to the index. The current index value is then compared to the index limit to determine if another loop should be performed.

Syntax

```
NEXT
```

Example

```
>1  REM EXAMPLE PROGRAM
>5  E=0 : C=10 : D=2
>10 FOR A=E TO C STEP D
>20 PRINT A
>30 NEXT A
>40 END
>RUN
```

```
0
2
4
6
8
10
```

```
READY
>
```

```
>NEW
>1  REM EXAMPLE PROGRAM
>10 FOR I = 0 TO 8 STEP 2
>20 PRINT I
>30 NEXT I
>40 END
>RUN
```

```
0
2
4
6
8
```

ONDF1

Use the ONDF1 statement to enable or disable the DF1 packet interrupt capability. (ONDF1 is equivalent to CALL 16, page 12 -10). You process the packet in an interrupt routine. Input the line number you want the program to jump to when a PRT2 receives a valid DF1 packet after the ONDF1 statement. Once you enable the DF1 packet interrupt, the BASIC module processor checks the port PRT2 receive buffer for a DF1 packet at the end of each line of BASIC. Execute a RETI (page 11 -33) within the interrupt routine to return to the point in the program before the interrupt occurred. To disable the DF1 packet interrupt input line number 0 after the DF1 statement. This statement has no effect if the DF1 protocol is not enabled (see CALL 108, page 13 -38) and JW4 (page 1 -6) is not in the correct position.

If the DF1 packet arrives due to a CALL 122 (page 13 -58) or 123 (page 13 -66), when ONDF1 is enabled, you receive the DF1 packet interrupt, but the DF1 packet has already been removed from the receive buffer.

When you enter Run mode ONDF1 is disabled until you enable it. ONDF1 is disabled when the BASIC module is in Command mode.

Syntax

```
ONDF1 ln num
```

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PACKET INTERRUPT
>20  ONDF1 800:REM LINE NUMBER OF START OF INTERRUPT ROUTINE
>800 (BEGINNING OF INTERRUPT ROUTINE)
.
. (PROCESS THE PACKET)
.
>850 RETI
```

ONERR

Use the ONERR statement to handle arithmetic errors, if they occur, during program execution. The ONERR statement only traps arithmetic overflow (value too large), arithmetic underflow (value too small), divide by zero, and bad argument errors. All other errors are not trapped and cause the BASIC module to enter the Command mode. If an arithmetic error occurs after you execute the ONERR statement, the module interpreter passes control to the line number (*ln num*) following the ONERR statement. You handle the error condition in a manner suitable to your application. The ONERR command does not trap bad data entered during an input instruction. This yields a TRY AGAIN message or EXTRA IGNORED message. For expanded ONERR functionality, refer to CALL 38 on page 12 -36.

Note: This statement did not work in the Series A module.

After you execute the ONERR statement, you can determine what type of error occurred by examining external memory location 257 (101H).

The error codes are:

- ERROR CODE = 10-DIVIDE BY ZERO
- ERROR CODE = 13-DIVIDE SYNTAX ERROR
- ERROR CODE = 20-ARITH. OVERFLOW
- ERROR CODE = 30-ARITH.UNDERFLOW
- ERROR CODE = 40-BAD ARGUMENT

You can examine this value by using an XBY(257) special function operator (page 9 -19). In addition, the line number can be determined by this BASIC line 256*XBY(69FDH)+XBY(69FEH).

Syntax

```
ONERR ln num
```

Example

```
>1  REM EXAMPLE PROGRAM
>10 ONERR 500
>20 FOR I = 5 TO 0 STEP -1
>30 PRINT 1/I
>40 NEXT I
>50 END
>500 PRINT "ERROR CODE WAS ",XBY(257)
>510 PRINT "AT LINE " (256*XBY(69FDH)+XBY(69FEH))
>510 END
READY
>RUN
.2
.25
.33333333
.5
1
ERROR CODE WAS 10
AT LINE 30
```

A GOTO statement (page 11 -14) can replace the END statement (page 11 -10) in this example to provide a method of user programmed error recovery.

ON-GOSUB

Use the ON-GOSUB statement to transfer control to the line(s) you specified with the GOSUB statement (page 11 -13) when the value of the expression following the ON statement is encountered in the BASIC program.

All comments that apply to GOSUB apply to the ON statement. If the *expr* after the ON is less than zero an `ERROR: BAD ARGUMENT` message is generated. If the *expr* after the ON is greater than the line number list following the GOSUB statement, an `ERROR: BAD SYNTAX` message is generated. The ON-GOSUB statement provides conditional branching options within the BASIC module program.

Important: Excessive nesting exceeds the limits of the control stack, generating an error, and causing the module to enter Command mode. For more information see control stack, page 8 -1.

Syntax

```
ON expr GOSUB ln num, ln num,...ln num
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 ON Q GOSUB 100,200,300
```

If Q is equal to 0, control is transferred to line number 100. If Q is equal to 1, control is transferred to line number 200. If Q is equal to 2, control is transferred to line number 300, and so on.

ONTIME

Use the ONTIME *expr, ln num* statement to compensate for the incompatibility between the timer/counters on the microprocessor and the BASIC module. Your BASIC module can process a line in milliseconds while the timer/counters on the microprocessor operate in microseconds. The ONTIME statement generates an interrupt every time the special function operator, TIME (page 9 -19), 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 that gives you seconds. This comparison is performed at the end (CR or :) of each line of BASIC. The interrupt forces a GOSUB (page 11 -13) to the line number after the expression in the ONTIME.

The ONTIME statement does not interrupt an input command or a call routine. Since the ONTIME statement uses the special function operator, TIME, you must execute the CLOCK1 statement (page 11 -5) for ONTIME to operate. If CLOCK1 is not executed the special function operator, TIME, does not increment.

To execute the ONTIME interrupt at a fraction of a second use the special function operator DBY(71) = X (page 9 -18) where X = 0 to 200. Each count represents a 5 millisecond time interval.

Syntax

```
ONTIME expr, ln num
```

Example

```
>10 TIME = 0
>15 DBY(71) = 0
>20 CLOCK1
>30 ONTIME 2,100
>40 DO
>50 WHILE TIME < 7
>60 CLOCK0
>70 END
>100 PRINT "TIMER INTERRUPT AT - ",TIME, " SECONDS"
>110 ONTIME TIME+2,100
>120 RETI

READY
>RUN

TIMER INTERRUPT AT - 2.01 SECONDS
TIMER INTERRUPT AT - 4.005 SECONDS
TIMER INTERRUPT AT - 6.015 SECONDS
```

The time printed out is .01 seconds later than the time supposed to be printed. The terminal used in the example operates at 19200 bit/s causing a .01 second delay in printing

ON-GOTO

Use the ON-GOTO statement to transfer control to the line(s) you specified with the GOTO statement (page 11 -14) when the value of the expression following the ON statement is encountered in the BASIC program.

All comments that apply to GOTO apply to the ON statement. If the *expr* after the ON is less than zero, an ERROR: BAD ARGUMENT message is generated and the BASIC module enters Command mode. If the *expr* after the ON is greater than the line number list following the GOTO statement, an ERROR: BAD SYNTAX message is generated. The ON-GOTO statement provides conditional branching options within the BASIC module program.

Syntax

```
ON expr GOTO ln num
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 ON Q GOTO 100,200,300
```

Control is transferred to line 100 if Q is equal to 0 and then to line 200 if Q is equal to 1. If Q is equal to 2, control is transferred to line number 300, and so on.

PH0. and PH1.

Use the PH0. and PH1. statements to direct the BASIC module to print a number in hexadecimal format to the console device. These statements function in the same way as the PRINT statement (page 11 -29) except that the values printed are in a hexadecimal format. The PH0. statement suppresses two leading zeros if the number is less than 255 (0FFH). The PH1. statement prints out four hexadecimal digits.

The character H always prints after the number when you use PH0. or PH1. to direct an output. The values are truncated integers. If the number is not within the range of valid integers (ex. between 0 and 65535 [0FFFFH] inclusive), the BASIC module defaults to the normal mode of print. If this happens an H does not print out after the value. Since you enter integers in either decimal or hexadecimal form, you can use the statements PRINT, PH0., and PH1. 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.

Important: You must ensure that buffer space is available anytime that you are printing data out of the serial port using hardware handshaking or software handshaking (XON/XOFF). Failure to do so causes the BASIC program to stop executing while awaiting buffer space. When space is available in the buffer, the BASIC module resumes execution from the point where it left off. The transmit buffer of each port is capable of holding 256 characters. See also the descriptions of CALLs 36 (page 12 -35), 37 (page 12 -35), 95 (page 13 -32), and 96 (page 13 -33).

Use the PH0# and PH1# operators to print to port PRT2 and the PH0@ and PH1@ operators to read port PRT1.

Syntax

<i>PH0.number or expression</i>	<i>PH1.number or expression</i>
<i>PH0.#number or expression</i>	<i>PH1.#number or expression</i>
<i>PH.0@number or expression</i>	<i>PH1.@number or expression</i>

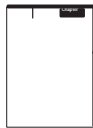
Example

```
>PH0.2*2
04H
>PH1.2*2
0004H
>PH0.100
64H
>PH0.1000
3E8H
>PH1.1000
03E8H
>PH1.3E8
3.0 E+8
```

POP

Use the POP statement to remove values from the BASIC module argument stack. The value at the top of the argument stack is assigned to the variable following the POP statement and the argument stack is popped (decrements by 6 bytes). You can place values in the stack using the PUSH statement (page 11 -30).

Important: If a POP statement executes and no number is in the argument stack, an A-stack error occurs and the BASIC module enters Command mode.



You can pop more than one variable off the argument stack using a single POP statement with multiple variables (*var; var; var*). You must follow each expression with a comma. The PUSH and POP statements accept dimensioned variables A(4) and S1(12) as well as scalar variables. This is useful when using call routines (see Chapters 12 and 13) in which large amounts of data must be pushed or popped.

You can use the PUSH and POP statements to minimize GLOBAL variable problems. These are caused by the main program and main program subroutines using the same variable names. If you cannot use the same variables in a subroutine as in the main program, you can re-assign a number of variables (example: A=Q) before a GOSUB statement (page 11 -13) is executed. If you reserve some variable names just for subroutines (S1, S2) and pass variables on the stack, you can avoid GLOBAL variable problems in the BASIC module.

Syntax

```
POP  var, var, .....var
```

Example

```
>1  REM EXAMPLE PROGRAM  
>40 FOR I=1 TO 64  
>50 PUSH I  
>60 CALL 10  
>70 POP A(I)  
>80 NEXT I
```

PRINT

Use the PRINT statement to direct the BASIC module to output a value to the console device. You may print the value of expressions, strings, literal values, variables or text strings. You may combine the various forms in the print list by separating them with commas. If the list is terminated with a comma, the carriage return/line feed is suppressed. P. is a shorthand notation for PRINT. Values are printed next to one another with two intervening blanks. A PRINT statement with no arguments sends a carriage return/line feed sequence to the console device.

Important: The BASIC interpreter terminates the printing of a string if it encounters a null (0), or CR (13) character. If you want to print strings containing these values, print the characters individually inside of a loop construct. To suppress the CR LF in the PRINT instruction, use a trailing comma. Example: print A,

Important: You must ensure that buffer space is available anytime that you are printing data out of the serial port using hardware handshaking or software handshaking (XON/XOFF). Failure to do so causes the BASIC program to stop executing while awaiting buffer space. When space is available in the buffer, the BASIC module resumes execution from the point where it left off. The output buffer of each port is capable of holding 256 characters. See descriptions of CALLs 36 (page 12 -35), 37 (page 12 -35), 95 (page 13 -32), and 96 (page 13 -33) for more information.

The symbols @ and # can be used to direct the print output to ports PRT1 and PRT2 respectively.

Syntax

Statement variation	Description
PRINT	output a value to the console device
PRINT CR	output a carriage return without a line feed
PRINT SPC()	output a specified number of spaces
PRINT TAB()	output a specified number of tab characters
PRINT USING(Fx)	output all numeric values in scientific notation. The x represents the total number of digits of the mantissa that are displayed. One digit is displayed before the decimal point. The value of x is a minimum of three and a maximum of eight. The value displayed is adjusted according to these limits.
PRINT USING(#. #)	output all numeric values in decimal notation according to the format specified
PRINT USING(0)	restore the default print mode if the mode was altered by the PRINT USING(Fx) expression, or by the PRINT USING(#. #) expression.

Example

```
>PRINT 10*10,3*3
100 9
```

PUSH

Use the PUSH statement to place the arithmetic expression or expressions in the BASIC module argument stack. This statement evaluates the arithmetic expression, or expressions, following the PUSH statement and then places them in sequence on the argument stack. Each variable PUSHed increments the A-stack by 6 bytes.



The PUSH and POP statements provide a simple means of passing parameters to call routines (see Chapters 12 and 13). In addition, the PUSH and POP statements are used to pass parameters to BASIC subroutines and to swap variables. The last value PUSHed onto the argument stack is the first value POPped off the argument stack.

You can push more than one expression onto the argument stack using a single PUSH statement with multiple expressions (*expr, expr, expr*). You must follow each expression with a comma. The last value pushed onto the argument stack is the last expression encountered in the push statement.

Important: The argument stack can hold up to 33 floating point numbers before overflowing.

Syntax

```
PUSH expr, expr, ... expr
```

Example

```
>1  REM EXAMPLE PROGRAM
>10 A = 10
>20 C = 20
>30 PRINT "A = ",A," AND C = " C
>40 PUSH A,C
>50 POP A,C
>60 PRINT "A = ",A," AND C = ",C
>70 END

READY
>RUN
A = 10 AND C = 20
A = 20 AND C = 10
READY
>NEW
>10 PUSH 0
>20 CALL 14
>30 POP W
>40 PRINT W
>50 END
READY
>RUN
0
```

READ

Use the READ statement to retrieve the expressions that you specified in the DATA statement (page 11 -6) and assign the value of the expression to the variable in the READ statement. The READ statement is always followed by one or more variables. If more than one variable follows a READ statement, separate them by a comma.

Every time a READ statement is encountered the next consecutive expression in the DATA statement is evaluated and assigned to the variable in the READ statement. You can place DATA statements anywhere within a program. They are not executed and do not cause an error.

DATA statements are chained together and appear as one large DATA statement. If at anytime all the data is read and you execute another READ statement, the program terminates and the message `ERROR: NO DATA - IN LINE XX` prints to the console device.

See also the DATA (page 11 -6) and RESTORE (page 11 -32) statements.

Syntax

```
READ
```

Example

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 READ A,C
>30 PRINT A,C
>40 NEXT I
>50 RESTORE
>60 READ A,C
>70 PRINT A,C
>80 DATA 10,20,10/2,20/2,SIN(PI),COS(PI)
```

```
READY
```

```
>RUN
```

```
10 20
```

```
5 10
```

```
0 -1
```

```
10 20
```

```
READY
```

```
>
```

REM

Use the REM command to specify a comment line in a BASIC program. Adding comment lines to a program makes the program easier to understand. Program lines that start with a REM command cannot be terminated with a colon (:). REM commands can be placed after a colon (:) in a program line. This allows you to place a comment on each line.

Important: REM commands add time to program execution. Use them selectively or place them at the end of the program where they do not affect program execution speed. Using REM commands in frequently called loops or subroutines slows the BASIC program execution.



Tip

After debugging the program save a fully commented copy on disk and remove comments from the executable program. PBASE provides a BDL Macro that automatically strips comments from a program.

Syntax

```
REM
```

Example

```
>10 REM THIS IS A COMMENT LINE  
>20 X=5 : REM THIS IS ALSO A COMMENT LINE
```

RESTORE

Use the RESTORE statement to reset the internal read pointer to the beginning of the data so that you may read it again.

See also the DATA (page 11 -6) and READ (page 11 -31) statements.

Syntax

```
RESTORE
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 FOR I = 1 TO 3  
>20 READ A,C  
>30 PRINT A,C  
>40 NEXT I  
>50 RESTORE  
>60 READ A,C  
>70 PRINT A,C  
>80 DATA 10,20,10/2,20/2,SIN(PI),COS(PI)  
READY  
>RUN  
10      20  
5       10  
0       -1  
10      20
```


RETI

Use the RETI statement to exit from an interrupt (ONDF1 (page 11 -22), ONTIME (page 11 -25), CALL 16 (page 12 -10,) or CALL 32 (page 12 -12) that is processed in a BASIC module program. The RETI statement functions the same as the RETURN statement (page 11 -34) except that it also clears a software interrupt flag so interrupts can again be acknowledged. If you do not execute the RETI statement in the interrupt procedure, all future interrupts are ignored.

Syntax

```
RETI
```

Example

```
>1  REM EXAMPLE PROGRAM
>10  TIME=0 : CLOCK1 : ONTIME 2, 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
```

RETURN

Use the RETURN statement to return control to the statement following the most recently executed GOSUB (page 11 -13). Use one return for each GOSUB to avoid overflowing the control stack. This means that a subroutine you call with the GOSUB statement can call another subroutine with another GOSUB statement.

Syntax

```
RETURN
```

Example

Simple Subroutine

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 5
>20 GOSUB 100
>30 NEXT I
>40 END
>100 PRINT I
>110 RETURN
READY
>RUN
1
2
3
4
5
```

Nested Subroutine

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 5
>20 GOSUB 100
>30 NEXT I
>40 END
>100 PRINT I,
>110 GOSUB 200
>120 RETURN
>200 PRINT I*I,
>210 GOSUB 300
>220 RETURN
>300 PRINT I*I*I
>310 RETURN
READY
>RUN
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
```

ST@

Use the ST@ statement to store BASIC module floating point numbers to a specified address. The expression following the ST@ statement specifies the address where you want the number stored in RAM. The ST@ statement takes the value on the top of the argument stack and stores it in RAM at the address location you specify by *expr*.

You can use this statement with CALL 77 (page 13 -6) to store variables to a protected area of memory. This protected area is not zeroed on power up or when you issue the RUN command (page 10 -19). Use the LD@ statement (page 11 -18) to retrieve the floating point number you stored with the ST@ statement.

Important: ST@ and LD@ are not used with any port designation.

Syntax

```
ST@ expr
```

Example

```
>PRINT MTOP
24575
>PRINT MTOP-10*6
24515
>PUSH 24515 : CALL 77
>PRINT MTOP
24515
>1  REM EXAMPLE PROGRAM
>5  DIM A(10),B(10)
>10 REM *** ARRAY SAVE ***
>20 FOR I = 0 TO 5
>30 A(I) = I+20
>40 PUSH A(I) : REM PUT NUMBER ON STACK
>50 ST@ 5FFFH-I*6
>60 NEXT I
>70 REM *** GET ARRAY ***
>80 FOR I = 0 TO 4
>90 LD@ 5FFFH-I*6
>100 POP B(I) : REM GET NUMBER FROM STACK
>110 PRINT B(I)
>120 NEXT I
READY
>RUN
20
21
22
23
24
READY
>PUSH 5FFFH : CALL 77
>PRINT MTOP
24575
```

STOP

Use the STOP statement to break program execution at specific points in a program. After a program is stopped you can display or modify variables. You can resume program execution with a CONT command (page 10 -3). The STOP statement allows for easy program debugging.

Note that the line number printed out after execution of the STOP statement is the line number following the STOP statement, not the line number that contains the STOP statement.

Syntax

```
STOP
```

Example

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 100
>20 PRINT I
>30 STOP
>40 NEXT I

READY
>RUN

  1
STOP - IN LINE  40
READY
>CONT

  2
STOP - IN LINE  40
READY
>CONT

  3
STOP - IN LINE  40
READY
>CONT

  4
STOP - IN LINE  40
READY
>
```

STRING

Use the `STRING` statement to allocate memory for strings. Initially, memory is not allocated for strings. If you attempt to define a string with a statement such as `LET $(1)="HELLO"` before memory is allocated for strings, an `ERROR: MEMORY ALLOCATION` message is generated. The first expression in the `STRING` statement is the total number of bytes you want to allocate for string storage. The second expression gives the maximum number of bytes in each string. The second value should not be larger than 254. These two numbers determine the total number of defined string variables.

You can only use one `STRING` statement in your program to allocate memory for all the strings you want to use in your program. When allocating memory for strings remember that the `STRING` statement itself has one overhead byte. As well, remember that BASIC uses one overhead byte per string declared within the `STRING` statement. The additional character for each string is allocated for the carriage return character that terminates the string. For example, the statement `STRING 100,10` allocates enough memory for nine 10-byte string variables, ranging from `$(0)` to `$(8)` and all of the 100 allocated bytes are used. Note that `$(0)` is a valid string in the BASIC module.

Important: If an ASCII null character is used within the string it acts as a marker denoting the end of a string.

Important: After memory is allocated for string storage, commands (ex. `NEW`, page 10 -10) and statements (ex. `CLEAR`, page 11 -2) cannot de-allocate this memory. Cycling power also cannot de-allocate this memory unless battery backup is disabled. You can de-allocate memory by executing a `STRING 0,0` statement. `STRING 0,0` allocates no memory to string variables.

Tip

Important: Define strings in your program first, unless you are executing a CALL 77 (page 13 -6). Then, execute the CALL 77 first and define your strings immediately after. The BASIC module executes the equivalent of a CLEAR every time you execute the STRING statement. This is necessary because string variables and numeric variables occupy the same external memory space. After the STRING statement executes, all variables and arrays are wiped out. Therefore perform string memory allocation early in a program (in the first statement if possible). If you re-allocate string memory you destroy all defined variables.

Important: The STRING statement turns off ONERR, CALL 38 and ONTIME. Make sure that the STRING statement is executed before the statements that enable interrupts.

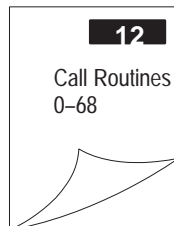
Syntax

```
STRING expr, expr
```

Example

```
>10 STRING 100,30  
>20 $(0) = "-----MONTHLY REPORT-----"  
>30 PRINT $(0)  
READY  
>RUN  
-----MONTHLY REPORT-----
```

What's Next?



Call Routines 0 – 68

What's in This Chapter?



Calls 0 – 68 are described here. Calls 69 – 127 are described in Chapter 13. Chapter 7 gives you an overview of how to use these calls within your BASIC program. Use these calls within your BASIC program or from the command line.

Important: CALL numbers above 127 are not valid and cause the BASIC module error—ERROR CALL ARGUMENT OUT OF RANGE.

CALL	Description	Page	CALL	Description	Page
0	reset module	12 -2	35	retrieve numeric input character from PRT2 port	12 -34
1	no operation	12 -2	36	get number of characters in PRT2 port buffers	12 -35
2	timed-block-transfer-read buffer	12 -2	37	clear PRT2 port buffers	12 -35
3	timed-block-transfer-write buffer	12 -3	38	expanded ONERR restart	12 -36
4	set block-transfer-write length	12 -4	39	3.3-Digit Signed, BCD to BASIC Floating Point	12 -38
5	set block-transfer-read length	12 -4	40	set wall clock time	12 -39
6	block-transfer-write buffer	12 -5	41	set wall clock date	12 -40
7	block-transfer-read buffer	12 -5	42	set wall clock day of week	12 -40
8	disable interrupts (no operation)	12 -6	43	date/time retrieve string	12 -41
9	enable interrupts (no operation)	12 -6	44	date retrieve numeric	12 -41
10	3-digit decimal BCD to BASIC floating point	12 -6	45	time retrieve string	12 -42
11	16-bit binary to BASIC floating point	12 -7	46	time retrieve numeric	12 -42
12	4-digit octal to BASIC floating point	12 -7	47	retrieve day of week string	12 -43
13	6-digit decimal BCD to BASIC floating point	12 -8	48	retrieve day of week numeric	12 -43
14	SLC 16-bit signed integer to BASIC floating point	12 -8	49	read remote DH-485 SLC data file	12 -44
15	SLC 16-bit unsigned integer to BASIC floating point	12 -9	50	write to remote DH-485 SLC data file	12 -50
16	enable/disable DF1 packet interrupt	12 -10	51	undefined	12 -58
17	4-digit BCD to BASIC floating point	12 -11	52	date retrieve string	12 -58
18	re-enable control C break function	12 -11	53	undefined	12 -58
19	disable the control C break function	12 -12	54	undefined	12 -58
20	BASIC floating point to 3-digit decimal BCD	12 -12	55	undefined	12 -58
21	BASIC floating point to 16-bit binary	12 -13	56	undefined	12 -58
22	BASIC floating point to 4-digit octal	12 -13	57	undefined	12 -58
23	BASIC floating point to 6-digit decimal BCD	12 -14	58	undefined	12 -58
24	BASIC floating point to SLC 16-bit signed integer	12 -15	59	undefined	12 -58
25	BASIC floating point to SLC 16-bit binary	12 -16	60	string repeat	12 -59
26	BASIC floating point to 3.3-digit BCD	12 -17	61	string append	12 -60
27	BASIC floating point to 4-digit BCD	12 -17	62	number to string conversion	12 -61
28	undefined	12 -18	63	string to number conversion	12 -62
29	read/write to PLC/SLC from module internal string	12 -18	64	find a string in a string	12 -63
30	PRT2 port support parameter set	12 -20	65	replace a string in a string	12 -64
31	display PRT2 port parameters	12 -21	66	insert a string in a string	12 -65
32	enable/disable processor interrupt	12 -22	67	delete a string from a string	12 -66
33	transfer data from PRT1/PRT2 to BTR buffer	12 -23	68	determine length of a string	12 -67
34	transfer data from BTW buffer to PRT1/PRT2	12 -29		what's next?	12 -67

CALL 0: Reset Module

This routine initiates a full reset. This is similar to a re-boot or pressing the reset button. The BASIC module reacts to this reset the same as it does when you turn on power to your I/O chassis backplane (page1 -13).

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 0
```

Example

```
> 10 CALL 0
```

CALL 1: No Operation

This routine does nothing. You return back to the main program.

CALL 2: Timed Block-Transfer-Read Buffer

Use this routine to send data to the PLC processor. CALL 2 transfers the block-transfer-read (BTR) buffer to the auxiliary processor on the BASIC module for use in the next BTR request from the PLC processor.

If a data transfer does not occur within 2 seconds the routine returns to your BASIC program without transferring data. BASIC execution halts until the BTR occurs or the call times out.

Whenever this call is active bit 2 (the sync BTR bit) of the PLC input image table is set. You can use this bit to trigger a BTR rung. If you are using CALLs 33, 34, 49, 50, 118, 122, or 123 you must use these bits to perform synchronous block transfers. Synchronous block transfer is only supported in PLC-5 processors; use the sync BTR bit with a PLC-5.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument is the status of the transfer. If CALL 2 times out before a successful block transfer is complete, the status code returned is 89 (59H). A time-out occurs if a previous CALL 2 initiated a block transfer read, and the BTR was not serviced by the PLC processor. CALL 2 does not return an error code if a mismatch occurs in block transfer length between the BASIC module and the block transfer read instruction.

- 0 = a successful transfer
- 89 = a transfer did not occur and the call timed out

Syntax

```
CALL 2  
POP status of transfer
```

Example

```
>10 CALL 2  
>20 POP X  
>30 IF X <> 0 PRINT "TRANSFER UNSUCCESSFUL"
```


CALL 3: Timed Block-Transfer-Write Buffer

Use this routine to receive data from the PLC processor. CALL 3 transfers the block-transfer-write (BTW) buffer to the auxiliary processor on the BASIC module for use in the next BTW request from the PLC processor. If a data transfer does not occur within 2 seconds the routine returns to your BASIC program with no new data. BASIC execution halts until the BTW occurs or the call times out.

Whenever this call is active bit 1 (the sync BTW bit) of the PLC input image table is set. You can use this bit to trigger a BTR rung. If you are using CALLs 33, 34, 49, 50, 118, 122, or 123 you must use these bits to perform synchronous block transfers. Synchronous block transfer is only supported in PLC-5 processors. Only use the sync BTW bit with a PLC-5 processors.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument is the status of the transfer.

- 0 = a successful transfer
- 92 (5CH) = mismatch in block transfer length between the BASIC module and the block-transfer-write instruction.
- 94 (5EH) = call timed out before a successful block transfer completed

Syntax

```
CALL 3  
POP status of transfer
```

Example

```
>10 CALL 3  
>20 POP X  
>30 IF X <> 0 PRINT "TRANSFER UNSUCCESSFUL"
```

CALL 4: Set Block-Transfer-Write Length

Use this routine to set the number of words (1-64) to transfer from the PLC processor to the BASIC module. The ladder logic program block-transfer length must match the set value.

Important: Only use CALL 4 in your program once to set the block-transfer-write block length.

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the number of words to BTW. If you do not use CALL 4 in your program, the default block-transfer length is 5 words. This must match the length of the PLC BTW instruction in the PLC ladder logic.

Syntax

```
PUSH number of words to BTW  
CALL 4
```

Example

```
>10 PUSH 10  
>20 CALL 4
```

CALL 5: Set Block-Transfer-Read Length

Use this routine to set the number of words (1-64) to transfer from the BASIC module to the PLC processor. The ladder logic program block-transfer length must match the set value.

Important: Only use CALL 5 in your program once to set the block-transfer-read block length.

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the number of words to BTR. If you do not use CALL 5 in your program, the default block-transfer length is 5 words. This must match the length of the PLC BTR instruction in the PLC ladder logic.

Syntax

```
PUSH number of words to BTR  
CALL 5
```

Example

```
>10 PUSH 10  
>20 CALL 5
```

CALL 6: Block-Transfer-Write Buffer

Use this routine to receive data from the PLC processor. CALL 6 transfers the block-transfer-write (BTW) buffer to the auxiliary processor on the BASIC module for use in the next BTW request from the PLC processor. This routine halts BASIC execution until a block-transfer write occurs.

Whenever this call is active bit 1 (the sync BTW bit) of the PLC input image table is set. You can use this bit to trigger a BTR rung. If you are using CALLs 33, 34, 49, 50, 118, 122, or 123 you must use these bits to perform synchronous block transfers. Synchronous block transfer is only supported in PLC-5 processors. Only use the sync BTW bit with a PLC-5 processors.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 6
```

Example

```
> 10 CALL 6
```

CALL 7: Block-Transfer-Read Buffer

Use this routine to send data to the PLC processor. CALL 7 transfers the block-transfer-read (BTR) buffer to the auxiliary processor on the BASIC module for use in the next BTR request from the PLC processor. This routine halts BASIC execution until a block-transfer read occurs.

Whenever this call is active bit 2 (the sync BTR bit) of the PLC input image table is set. You can use this bit to trigger a BTR rung. If you are using CALLs 33, 34, 49, 50, 118, 122, or 123 you must use these bits to perform synchronous block transfers. Synchronous block transfer is only supported in PLC-5 processors. Only use the sync BTR bit with a PLC-5 processors.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 7
```

Example

```
> 10 CALL 7
```

**CALL 8: Disable Interrupts
(No Operation)**

This routine was used during EPROM programming in the 1771-DB, Series A BASIC module. If you initiate this call with a 1771-DB, Series B BASIC module, nothing happens. The call simply returns to the main program.

**CALL 9: Enable Interrupts
(No Operation)**

This routine was used during EPROM programming in the 1771-DB, Series A BASIC module. If you initiate this call with a 1771-DB, Series B BASIC module, nothing happens. The call simply returns to the main program.

**CALL 10:
3-Digit Signed, Fixed
Decimal BCD to
BASIC Floating Point**



Use this routine to convert 3-digit BCD from PLC processor to BASIC floating point. See also CALL 20.

See Chapter 8 for more information.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the number (1 to 64) of the word in the block-transfer-write buffer that you want to convert from 3-digit BCD to BASIC format. The output argument is the value converted into BASIC floating point format. The sign bit is bit number 16.

Syntax

```
PUSH number of word (1 – 64) to be converted  
CALL 10  
POP converted value
```

Example

```
> 10 PUSH X  
> 20 CALL 10  
> 30 POP Y
```

CALL 11: 16-Bit Binary to BASIC Floating Point



Use this routine to convert 16-bit binary from PLC processor to BASIC floating point. See also CALL 21.

See Chapter 8 for more information.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the number (1 to 64) of the word in the write-data-transfer buffer you want to convert from 16-bit binary to BASIC format. The output argument is the value converted into BASIC floating point format. There are no sign or error bits decoded.

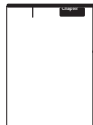
Syntax

```
PUSH number of word (1 – 64) to be converted  
CALL 11  
POP converted value
```

Example

```
> 10 PUSH X  
> 20 CALL 11  
> 30 POP Y
```

CALL 12: 4-Digit Signed Octal to BASIC Floating Point



Use this routine to convert 4-bit octal from the PLC processor to BASIC floating point. See also CALL 22.

See Chapter 8 for more information.

Input and Output Arguments

The input argument is the number (1 to 64) of the word in the processor block-transfer buffer you want to convert from 4-digit signed octal to BASIC floating point format. This 12-bit format has a maximum value of ± 7777 octal. The output argument is the value converted into BASIC floating point format. The sign bit is bit number 16.

Syntax

```
PUSH number of word ( 1– 64) to be converted  
CALL 12  
POP converted value
```

Example

```
> 10 PUSH X  
> 20 CALL 12  
> 30 POP Y
```

CALL 13:
**6-Digit Signed, Fixed
Decimal BCD to
BASIC Floating Point**



Use this routine to convert 6-digit BCD from the PLC processor to BASIC floating point. See also CALL 23.

See Chapter 8 for more information.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the number (1-64) of the first word (6-digit BCD is sent to the BASIC module in two processor words) of the write-block-transfer buffer you want to convert from 6-digit, signed, fixed decimal BCD to BASIC floating point. The maximum values allowed are ± 999999 . The output argument is the value converted into BASIC floating point format. The sign bit is bit number 16.

Syntax

```
PUSH number of word (1 – 64) to be converted  
CALL 13  
POP converted value
```

Example

```
> 10 PUSH X  
> 20 CALL 13  
> 30 POP Y
```

CALL 14:
**SLC 16-Bit Signed Integer to
BASIC Floating Point**



Use this call to convert an SLC 16-bit signed integer number from an SLC processor to a BASIC floating-point number. This call is used to convert SLC signed integer numbers to BASIC floating point. Use it with DH-485 related calls. See also CALL 24.

See Chapter 8 for more information.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the address number (100 to 139) of the word in the BASIC module input buffer DH-485 common interface file you want to convert. The output argument is the value converted into BASIC floating point format.

Syntax

```
PUSH number of word to be converted  
CALL 14  
POP converted value
```

Example

```
>1 REM EXAMPLE PROGRAM  
>20 PUSH 101:REM CONVERT 101ST BYTE OF BASIC INPUT BUFFER  
>30 CALL 14 : REM DO 16-BIT SIGNED TO F.P. CONVERSION  
>40 POP W : REM GET CONVERTED VALUE  
>50 PRINT W
```

CALL 15: SLC 16-Bit Unsigned Integer to BASIC Floating Point



Use CALL 15 to convert an SLC 16-bit unsigned integer (or 16-bit binary) number from an SLC processor to a BASIC module floating-point number. This call is used to convert SLC unsigned integer numbers to BASIC floating point. Use it with DH-485 related calls. See also CALL 25.

See Chapter 8 for more information.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the address number (100 to 139) of the word in the BASIC module input buffer DH-485 common interface file you want to convert. The output argument is the value converted into BASIC floating point format.

Syntax

```
PUSH number of word to be converted  
CALL 15  
POP converted value
```

Example

```
>50 PUSH 120:REM CONVERT 120TH BYTE OF BASIC INPUT BUFFER  
>60 CALL 15 :REM DO 16-BIT INTEGER TO F.P. CONVERSION  
>70 POP L(9) :REM GET CONVERTED VALUE-store in array L(9)
```

CALL 16: Enable/Disable DF1 Packet Interrupt

Use this routine to enable or disable the DF1 packet interrupt capability. This call has the same functionality as the ONDF1 statement (page 11 -22). You process the DF1 packet within an interrupt routine. To return to the point in the program before the interrupt occurred, execute a RETI (page 11 -33) within the routine. Once you enable this call the processor checks the PRT2 port for a received DF1 message at the end of each line of BASIC code.

DF1 packets are disabled when the BASIC module is in Command mode. When you enter Run mode CALL 16 is disabled until you enable it. You must re-execute CALL 16 every time your enter Run mode.

If the DF1 packet arrives in PRT2 due to a CALL 122 (page 13 -58) or CALL 123 (page 13 -66) you receive the DF1 packet interrupt when you enable CALL 16. However, the DF1 packet is not in BASIC module input buffer. It has already been removed.

This command has no effect if you have not set JW4 (page 1 -6) properly and enabled the DF1 protocol (see CALL 108, page 13 -38).

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the beginning BASIC line number of the interrupt routine. The program jumps to this line number when the PRT2 port buffer receives a valid DF1 packet. A line number of 0 disables the DF1 packet interrupt.

Syntax

```
PUSH beginning line number of interrupt routine  
CALL 16  
RETI
```

Example

```
>20  PUSH 800: REM LINE NUMBER OF START OF INTERRUPT ROUTINE  
>30  CALL 16  
>800 (BEGINNING OF INTERRUPT ROUTINE)  
  
:    (PROCESS THE PACKET)  
  
>850 RETI
```


CALL 17: 4-Digit BCD to BASIC Floating Point



Use this routine to convert 4-digit BCD from the PLC processor to BASIC floating point. See also CALL 27.

See Chapter 8 for more information.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the number (1-64) of the word in the block-transfer-write buffer you want to convert from 4-digit BCD to BASIC floating point format. The maximum value allowed is 0-9999. The output argument is the value converted into BASIC floating point format.



Syntax













```
PUSH number of word (1 – 64) to be converted
CALL 17
POP converted value
```

Example

```
> 10 PUSH X
> 20 CALL 17
> 30 POP Y
```

CALL 18: Re-Enable Control C Break Function

Use CALL 18 to re-enable the  +  break function for LIST (page 10 -9) and RUN (page 10 -19) operations. Execute CALL 18 in a BASIC program or from Command mode.

Important: When  +  is disabled, you are unable to stop program execution through a BASIC command. Cycling power re-enables  +  checking until the program once again disables  + . To stop program execution, you must cycle power and press  +  before you execute the line that disables  + . You can change JW4 (page 1 -6) to the default position to stop a program that has  +  disabled.

Input and Output Arguments

This routine has no input or output arguments.





Syntax

```
CALL 18
```

Example

```
> 90 CALL 18
```

CALL 19: Disable the Control C Break Function

Use CALL 19 to disable the  +  break function for LIST (page 10 -9) and RUN (page 10 -19) operations. Execute CALL 19 in a BASIC program or from Command mode. Cycling power returns the  +  function to normal operation if you disable it from the Command mode.

Important:  +  is enabled by default.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 19
```

Example

```
> 90 CALL 19
```

CALL 20: BASIC Floating Point to 3-Digit, Signed, Fixed Decimal BCD



Use this routine to convert BASIC floating point to 3-digit PLC BCD number. See also CALL 10.

See Chapter 8 for more information.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input argument is the variable with a value in the range of ± 999 that you want to convert to signed 3-digit BCD format. The second input argument is the number of the word (1–64) to receive the converted value in the block-transfer-read buffer. The sign bit is bit number 16.

Syntax

```
PUSH data to be converted  
PUSH word location (1–64) to receive value  
CALL 20
```

Example

```
>20 PUSH W :REM DATA TO BE CONVERTED  
>30 PUSH 6 :REM WORD LOCATION TO GET DATA  
>40 CALL 20
```

CALL 21: BASIC Floating Point to 16-Bit Binary



Use this routine to convert BASIC floating point to 16-bit binary PLC number. This routine takes a value between 0 and 65535 and converts it to its binary representative and stores it in the block-transfer-read buffer in one word. See also CALL 11.

See Chapter 8 for more information.

Input and Output Arguments

This routine has two input and no output arguments. The first input value is the data or variable you want to convert to 16-bit binary. The second input value is the number of the word (1–64) to receive the converted value in the block-transfer-read buffer.

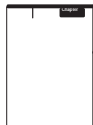
Syntax

```
PUSH value to be converted
PUSH word location (1–64) that receives the value
CALL 21
```

Example

```
>50 PUSH T :REM THE VALUE TO BE CONVERTED TO 16 BINARY
>60 PUSH 3 :REM WORD 3 IN THE BTR BUFFER GETS THE VALUE T
>70 CALL 21 :REM DO THE CONVERSION
```

CALL 22: BASIC Floating Point to 4-Digit, Signed Octal



Use this routine to convert a value from BASIC format to a four-digit, signed, octal PLC value. See also CALL 12.

See Chapter 8 for more information.

Input and Output Arguments

This routine has two input and no output arguments. The first input value is the data ($\pm 7777_8$) or variable you want to convert to 4-digit, signed octal. The second input value is the number of the word (1–64) to receive the converted value in the block-transfer-read buffer. The sign bit is bit number 16.

Syntax

```
PUSH value to be converted
PUSH word location (1–64) that receives the value
CALL 22
```

Example

```
>50 PUSH H :REM THE VALUE TO BE CONVERTED TO 4-DIGIT
          SIGNED OCTAL
>60 PUSH 3 :REM WORD 3 IN THE BTR BUFFER GETS THE VALUE H
>70 CALL 22 :REM DO THE CONVERSION
```

CALL 23:
BASIC Floating Point to
6-Digit, Signed, Fixed
Decimal BCD



This routine converts a value from BASIC floating point format to 6-digit, signed, PLC BCD number in a 2 word format and places the converted value in the block-transfer-read buffer. See also CALL 13.

See Chapter 8 for more information.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input value is the data or variable. The second input value is the number of the word (1-64) to receive the converted value in the block-transfer-read buffer. The sign bit is bit number 16.

Syntax

```
PUSH value to be converted  
PUSH word location (1-64) that receives the value  
CALL 23
```

Example

```
>10 W=567321  
>20 PUSH W  
>30 PUSH 3  
>40 CALL 23
```

CALL 24: BASIC Floating Point to SLC 16-Bit Signed Integer



Use CALL 24 to convert a BASIC floating-point number to an SLC 16-bit signed integer SLC number and place result in the BASIC module output buffer, DH-485 common interface file. This call is used with DH-485 calls for BASIC module to SLC data conversion. See also CALL 14.

See Chapter 8 for more information.

The fractional part of the BASIC floating-point value is truncated. If the BASIC floating-point value is less than -32768, the value placed in the BASIC module output buffer is -32768. If the BASIC floating-point value is greater than +32767, the value placed in the BASIC module output buffer is +32767. You are responsible for checking the range of the number before conversion.

Input and Output Arguments

This routine has two input and no output arguments. The first input value is the data variable you want to convert to 16-bit integer. The second input value is the address number (100 to 139) of the word that receives the converted value in the BASIC module output buffer.

Syntax

```
PUSH value to be converted  
PUSH word location (100-139) that receives the converted value  
CALL 24
```

Example

```
>10 W = 17  
>40 PUSH W : REM THE VALUE TO BE CONVERTED  
>50 PUSH 120 : REM WORD 120 OF BASIC OUTPUT BUFFER GETS W.  
>60 CALL 24 : REM DO THE F.P. TO 16-BIT SIGNED CONVERSION
```

CALL 25: BASIC Floating-Point to SLC 16-Bit Binary



Use CALL 25 to convert a BASIC module floating-point value between 0 and 65535 to its 16-bit binary (or unsigned integer) SLC number and store the result in the BASIC module output buffer DH-485 common interface file. This call is used with DH-485 calls for BASIC module to SLC data conversion. See also CALL 15.

See Chapter 8 for more information.

The fractional part of the BASIC module floating-point value is truncated. If the BASIC module floating-point value is less than 0, then the value placed in the output buffer is 0. If the value is greater than +65535, then the value placed in the output buffer is +65535. You are responsible for checking the range of the number before conversion.

Input and Output Arguments

This routine has two input and no output arguments. The first input value is the data you want to convert to 16-bit binary. The second input value is the address number (100 to 139) of the word in the BASIC module output buffer to receive the converted value.

Syntax

```
PUSH value to be converted  
PUSH word number (100–139) of BASIC module output buffer  
CALL 25
```

Example

```
>40 PUSH 0A5H : REM THE VALUE TO BE CONVERTED  
>50 PUSH 110 : REM WORD 110 OF BASIC MODULE OUTPUT BUFFER  
>60 CALL 25 : REM DO F.P. TO 16-BIT BINARY CONVERSION
```

CALL 26:
BASIC Floating Point to
3.3-Digit Signed BCD



Use this routine to convert a variable in BASIC floating point format to a signed, 6-digit, fixed decimal point PLC number and store it in 2 words in the block-transfer-read buffer. See also CALL 39.

See Chapter 8 for more information.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input value is the data (± 999.999) or variable you want to convert to 3.3-digit, signed BCD. The second input value is the number of the first word (1–64) to receive the converted value in the BTR buffer. The sign bit number is bit 16.

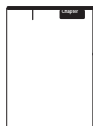
Syntax

```
PUSH value to be converted
PUSH word number (1–64) of BASIC module BTR buffer
CALL 26
```

Example

```
>50 PUSH S      :REM VALUE TO BE CONVERTED
>60 PUSH 3      :REM WORD 3 IN THE BTR BUFFER GETS THE VALUES
>70 CALL 26     :REM DO THE CONVERSION
```

CALL 27:
BASIC Floating Point to
4-Digit BCD



Use this routine to convert a value in BASIC floating point format to a 4-digit, unsigned BCD PLC value and place it in the block-transfer-read buffer. See also CALL 17.

See Chapter 8 for more information.

Input and Output Arguments

This routine has two input and no output arguments. The first input value is the data (0-9999) or variable. The second input value is the number of the word (1–64) to receive the converted value in the block-transfer-read buffer.

Syntax

```
PUSH value to be converted
PUSH word number (1–64) of BASIC module BTR buffer
CALL 27
```

Example

```
>20 PUSH B      : REM THE VALUE TO BE CONVERTED TO 4-DIGIT BCD
>30 PUSH 7      :REM WORD 7 IN THE BTR BUFFER GETS THE VALUE B
>40 CALL 27     :REM DO THE CONVERSION
```

CALL 28

Undefined. If you execute an undefined call, you receive the error message, "ERROR-UNSUPPORTED CALL."

CALL 29: Read/Write to a PLC/SLC Processor from the BASIC Module Internal String

Use CALL 29 in conjunction with CALL 122 (page 13 -58) or CALL 123 (page 13 -66) to communicate between remote PLC processors and the BASIC module internal string without local PLC processor interaction. You can also use CALL 29, in conjunction with CALL 49 (page 12 -44) or CALL 50 (page 12 -50), to communicate between remote SLC processors and the BASIC module internal string (see String Data Types, page 8 -2) without local PLC processor interaction. You must execute CALL 49, 50, 122, or 123 within the BASIC program before CALL 29.

CALL 29 is active when the internal string is the only choice in CALLs 49, 50, 122, or 123. In this situation, it is not practical to use the PLC block transfer words to begin the transfer and to pass the status. The PLC processor does not need to be involved. If you choose a PLC BTR or BTW information, the local PLC processor controls the transfer with the I/O bits. In this instance, when you attempt CALL 29, you receive a status of 255.

When you execute CALL 29, the transfer is attempted. If you have not executed the selected call (49, 50, 122, or 123) before executing CALL 29 you receive a status of 1 in the output argument. When you execute CALL 29 successfully, the value in the first character of the string (transaction number) increments to designate that the transfer occurred. The range of this character is 0-255.

Unlike CALLs 49, 50, 122, and 123, CALL 29 does not require PLC bit handshaking at the end of the command when using an SLC file as a source or destination.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the call you want to activate (CALL 49, 50, 122, or 123). The output argument is the status of the transaction:

- 0 = successful completion
- 1 = chosen call (49, 50, 122, or 123) is not active
- 51 = PLC/SLC command already in progress
- 57 = bad input argument
- 255 = PLC BTW or BTR is chosen for CALL 49, 50, 122, or 123 and CALL 29 is ignored
- all other codes are identical to CALL 90/92 (see page 13 -18 or 13 -26)

Syntax

PUSH 49, 50, 122, or 123 for the CALL you want to activate
CALL 29
POP status of transaction

Example

CALL 122 must be enabled with internal string only prior to executing CALL 29 in this example. Upon execution of CALL 29, an attempt is made to transfer one element from integer file 10, starting at element 0 of the PLC-5 processor at node 3, to the internal string \$(1) of the BASIC module.

```
>5  STRING 1000, 100
>10  REM EXECUTE DF1 PLC REMOTE READ FROM INTERNAL
>20  REM STRING WITH NO PLC INTERVENTION
>21  REM SET UP CALL 122
>25  PUSH 5, 3, 10, ASC(N), 0, 10, 10, 1, 1, 1: CALL 122
>27  POP STATUS
>30  PUSH 122
>40  CALL 29
>50  POP S
>60  IF (S=1) THEN PRINT "CALL 122 NOT ACTIVE"
>70  IF (S=255) THEN PRINT "PLC FILE CHOSEN FOR CALL 122"
>80  IF (S=0) THEN PRINT "SUCCESSFUL TRANSFER"
>90  IF (S<>0) THEN PRINT "UNSUCCESSFUL TRANSFER"
>100 END
```

CALL 30: PRT2 Port Support Parameter Set

Use this routine to set up the parameters for the PRT2 port. The parameters you set are the number of bits/word, parity enable or disable/even or odd, number of stop bits, and handshaking (software, hardware, or none). The default communication rate is 1200 bit/s (jumper selected) and the default start bit is 1 (fixed).

You can also use the MODE statement (page 11 -20) to set PRT2 port parameters.

Important: From the factory, software handshaking is enabled for PRT2. If you change this or any other PRT2 port parameter through a CALL 30 or MODE statement and a execute PROG1 or PROG2, then the configuration you selected with the CALL 30 or MODE statement is the power on default. On power up the receive and transmit buffers are cleared.

The 1771-DB Series A, Rev. E added a RTS control mode to CALL 30. This RTS control mode enabled the RTS signal and the RS-422 output pin on each character transmitted. The 1771-DB Series B automatically activates the RTS line any time a character is transmitted and deactivates RTS after the last character exits the port. This feature allows RS-422 multi-point or point-to-point to function properly when hardware handshaking is disabled.

Input and Output Arguments

This routine has six input and no output arguments. The input arguments are (in this order):

Parameter	Selections ^①
number of bits/word	5, 6, 7, 8
parity enable	0=None, 2=Even, 1=Odd
number of stop bits	1=1 Stop Bit, 2=2 Stop Bits, 3=1.5 Stop Bits
software handshaking	0=None, 1=XON-XOFF
hardware handshaking	0=CTS, DCD and DSR ignored 1=CTS, DCD and DSR used for handshaking
dummy argument (optional) ^②	ignored (Series A compatibility)

^①Default conditions are controlled by settings when PROG1 or PROG2 is executed

^②Dummy argument is for 1771-DB Series A compatibility. It is not required for 1771-DB Series B operation. If it is used, the 1771-DB Series B simply ignores it.

Syntax

```
PUSH number of bits/word
PUSH parity
PUSH number of stop bits
PUSH software handshaking
PUSH hardware handshaking
CALL 30
```

Example

```
>100 PUSH 8 :REM 8 BITS/WORD
>120 PUSH 0 :REM NO PARITY
>140 PUSH 1 :REM 1 STOP BIT
>160 PUSH 0 :REM NO SOFTWARE HANDSHAKING
>180 PUSH 0 :REM IGNORE CTS, DCD and DSR
>190 CALL 30 :REM SET UP PRT2 PORT
```

-or-

```
>100 PUSH 8, 0, 1, 0, 0:CALL 30
```


CALL 31: Display PRT2 Port Parameters

This routine displays the current PRT2 port configuration on the terminal. Enter CALL 31 from the Command mode.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 31 
```

Example

```
>CALL 31

1200 BAUD
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
```

Important: Software handshaking status is shown only if enabled.

CALL 32: Enable/Disable Processor Interrupt

Important: This call requires the BASIC module jumper JW5 to be in 16 point mode (page 1 -7).

Use CALL 32 to allow the PLC processor to interrupt the BASIC module. When enabled the BASIC module monitors PLC output bit 16 for a 0 to 1 transition at the end of every BASIC line and generates the interrupt if the bit is set. The PLC ladder logic should then clear the PLC output bit 16.

Interrupts are disabled when the BASIC module is in Command mode. When you enter Run mode CALL 32 is disabled until you enable it. You must re-execute CALL 32 every time you enter Run mode.

Input and Output Arguments

This routine has one input and no output arguments. The input argument is the beginning line number of the interrupt routine. The program jumps to this line number when PLC output bit 16 is set. The BASIC module detects this transition automatically and jumps to the interrupt routine. If you execute a RETI (page 11 -33) within the interrupt routine, you return to the point in the program before the interrupt occurred. Line number 0 disables the processor interrupt.

Syntax

```
PUSH beginning line number of interrupt routine  
CALL 32
```

Example

```
>1      REM EXAMPLE PROGRAM  
>10     REM ENABLE PROCESSOR INTERRUPTS  
>20     PUSH 1000  
>30     CALL 32  
>1000   (BEGINNING OF THE PROCESSOR INTERRUPT ROUTINE)  
:  
>1050   RETI
```

Sample Ladder Logic



CALL 33: Transfer Data from PRT1 or PRT2 to the BTR Buffer



Tip

Important: This call requires the BASIC module jumper JW5 to be in 16 point mode (page 1 -7).

Use CALL 33 to transfer data from the BASIC module ASCII ports directly to the BTR buffer and/or an internal string within the BASIC module.

This call is useful for reading bar code data and sending it to the PLC processor or an internal string. Once you set up and enable this operation, it is performed transparently in the background while the BASIC module executes a BASIC program in the foreground.

During data transfer, data is automatically transferred in 8-bit blocks from the receive buffer of the port you selected to the BTR buffer locations and/or BASIC internal string you selected for storage. The transfer occurs when the BASIC module detects the number of characters you specified in the receive buffer of the port or receives the user-defined delimiter in the port. You can store the data either low byte first, then high byte, or high byte first, then low byte within the 16-bit word of the destination. Data is transferred on word boundaries. If you transfer an odd number of bytes, the unused byte contains a zero.

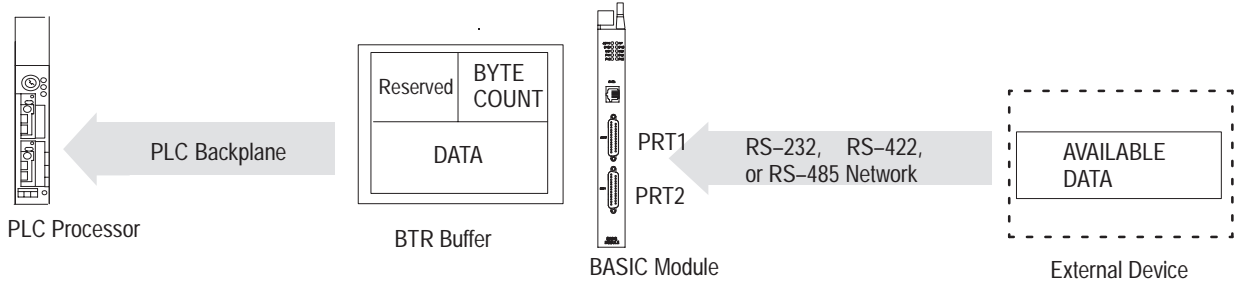
The byte swap selection (low byte first, then high byte, or high byte first, then low byte) of the last CALL 33 or CALL 34 you executed determines the data packing method for all the ports that CALL 33 enables.

The low byte of BTR word 1 contains the character count (byte count) of the data you transferred. If a delimiter is found, the byte count is expanded to include the first occurrence of the delimiter. BTR word 2 contains the first two characters of data.

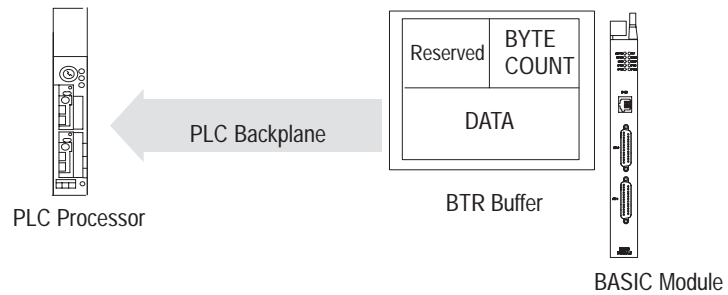
If you choose an internal string, the first character of the string contains the byte count. The second character of the internal string is a transaction number and increments to inform the BASIC module that new data is in the string. The value of this character wraps around from 255 to 0. The third character of the string contains the first data character.

Execute CALL 33 to set up the data transfer parameters. After you execute the call, the BASIC module gets data from the port and transfers it to the destination.

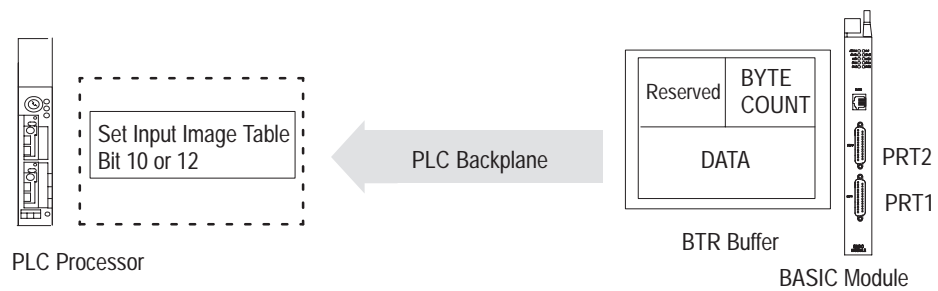
1. When data is available from the port, the BASIC module automatically transfers the data into the BTR buffer. The module also checks the same port for data at the end of each line of BASIC code.



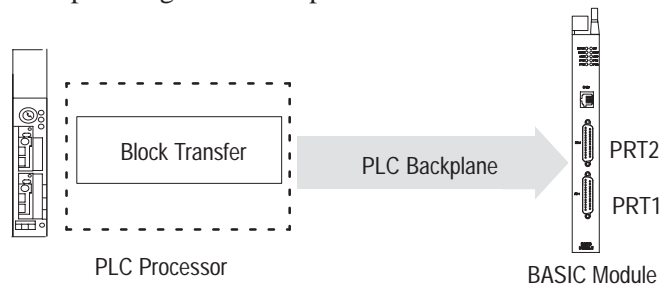
2. The BASIC module places the byte count of the valid data into the lower byte of BTR word 1 of the BTR buffer. The upper byte of BTR word 1 is reserved.



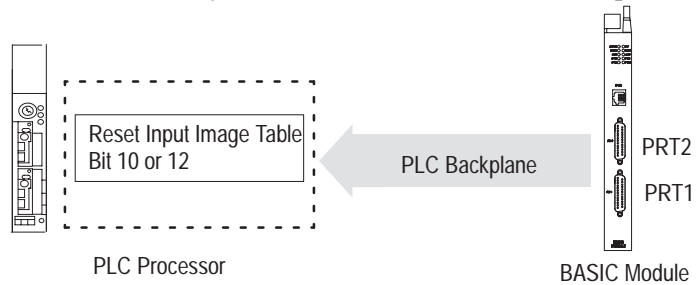
3. The BASIC module sets a bit in the PLC input image table to inform the PLC processor that valid data is available. Bit 10 indicates that data is available from PRT1 and bit 12 indicates that data is available from PRT2.



4. The ladder logic program of the PLC processor retrieves the data from the input image table and performs a block transfer



5. The BASIC module resets the bit in the PLC input image table on the same end of the scan cycle in which the block transfer is performed.



Data transfers continue until you re-execute the call for the port with different input parameters. If this occurs, the previous CALL 33 for the port is automatically disabled and the new CALL 33 takes effect. You cannot execute multiple CALL 33s for the same port in parallel. However, you can activate PRT1 and PRT2 simultaneously by issuing separate CALL 33s.

Input and Output Arguments

This routine has seven input arguments and one output argument.

Argument	Description	Page
input 1	source port of BASIC module	12 -26
input 2	maximum number of 8-bit characters you want to copy from the BASIC serial port to the destination file	12 -26
input 3	decimal value of the ASCII character delimiter	12 -26
input 4	selection of the destination buffer	12 -26
input 5	always 1	12 -27
input 6	the string number	12 -27
input 7	byte swap selection	12 -27
output 1	call status	12 -27

Input Argument One

The first input argument is the source port number (PRT1 or PRT2) of the BASIC module. A zero disables all previously active CALL 33 commands.

- 0 = disable CALL 33 for all active ports enabled by earlier CALL 33s
- 1 = PRT1 is source
- 2 = PRT2 is source

Input Argument Two

The second input argument is the maximum number of 8-bit characters you want to copy from the BASIC serial port to the destination file.

The destination of your data (input argument four) determines the maximum number of characters. These selections have a maximum number of characters of:

- PLC BTR = 126 (63 words)
- Internal string = *string size* – 3 (254 maximum characters)
 - first character is the byte count value
 - second character is the incremented transaction number
 - last character is the terminating character

If the port has acquired less than the maximum when it receives a delimiter character, it sends the packet including the delimiter to the BTR buffer.

The ladder logic can determine the amount of valid data transferred into the BTR buffer from the byte count placed into the lower byte of BTR word 1. If the data received exceeds the string length or BTR buffer size, the remaining data is truncated.

Input Argument Three

The third input argument is the decimal value of the ASCII character delimiter. You can choose any valid ASCII character. If you do not want a delimiter, enter a null value (0 decimal). The data is transferred to the destination buffer when the delimiter is received from the selected port regardless of the number of characters received.

Input Argument Four

The fourth input argument is the selection of the BTR buffer with or without the internal string or the internal string alone:

- 0 = BTR buffer
- 2 = BTR buffer and internal string
- 4 = internal string only

When transferring data to the internal string of the BASIC module, check your transaction number for string updates because there is no indication that data has been placed in the internal string. Your BASIC program must check the transaction number to verify that the data was updated.

Input Argument Five

The fifth input argument should always be 1.

Input Argument Six

The sixth input argument is the string number. If the fourth input argument does not select internal string usage, the value of this input argument is ignored, but you must still PUSH the argument.

Input Argument Seven

The seventh input argument is the byte swap selection:

- 0 = data bytes transferred from the BASIC port are *not* swapped when passed to the destination. The data packing order is low byte first, then high byte per word. The low byte of the first word in the destination file contains the byte count.
- 1 = data bytes transferred from the BASIC port are swapped when passed to the destination. The data packing order is high byte first, then low byte per word. Swapping does not affect the first word. The low byte of the first word still contains the byte count.

The last CALL 33 or CALL 34 executed determines the byte swap option for all active CALL 33 commands previously executed.

Output Argument One

The output argument is the status of the call. It has these values:

- 0 = successful setup
- 1 = disabled
- 2 = bad input parameter
- 3 = PRT2 is chosen but it is already enabled for DF1 protocol.
Transfer is not executed.
- 4 = string too small
- 5 = string is not dimensioned
- 6 = JW5 not in 16-point position

If data is being received into the serial port faster than the PLC processor is retrieving data, the input buffer of the port fills up. If you use handshaking between the port and the external device, no data is lost.

Syntax

PUSH *source port number*
PUSH *maximum number of characters to be transferred*
PUSH *decimal value of character delimiter*
PUSH *selection of BTR buffer and/or string*
PUSH 1
PUSH *string number*
PUSH *byte swap selection*
CALL 33
POP *CALL 33 status*

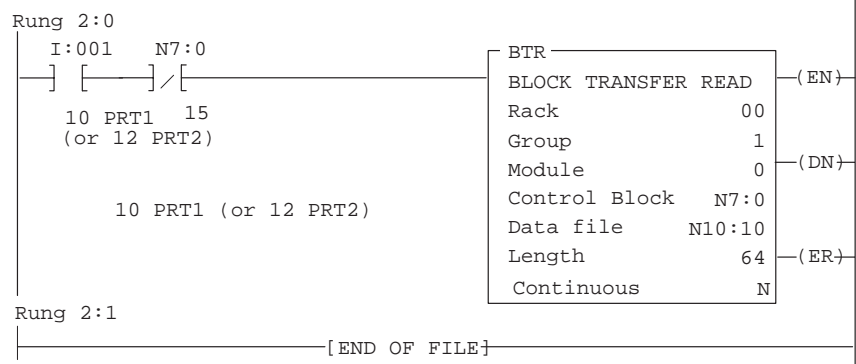
Example

```

>1  REM EXAMPLE PROGRAM
>05 PUSH 64 : CALL 5 : REM SET BLOCK TRANSFER READ
    LENGTH
>10 REM ENABLE CALL 33 INTERRUPTS
>20 PUSH 1 : REM PRT1 ACTIVE FOR CALL 33
>30 PUSH 10 : REM RECEIVING 10 BYTES OF DATA MAXIMUM
>40 PUSH 13 : REM <CR> USED AS TERMINATION CHARACTER
    (13 DECIMAL)
>50 PUSH 0 : REM SEND DATA BTR BUFFER
>60 PUSH 1 : REM OFFSET ALWAYS 1
>70 PUSH 0 : REM STRING NUMBER - NOT USED
>80 PUSH 1 : REM BYTE SWAPPING ENABLED
>90 CALL 33
>100 POP S : REM STATUS OF CALL 33 SETUP
>110 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 33 SETUP"
>120 END

```

Sample Ladder Logic



CALL 34: Transfer Data from the BTW buffer to PRT1 or PRT2



Important: This call requires the BASIC module jumper JW5 to be in 16 point mode (page 1 -7).

Use CALL 34 to transfer data from the BTW buffer file directly to the BASIC module serial port and/or to a string within the BASIC module.

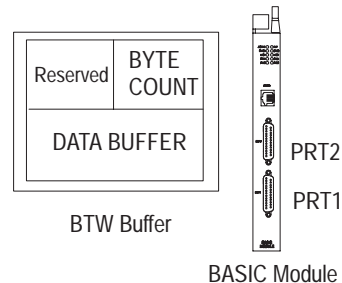
This call is useful for writing to a remote display device (operator interface) directly from the PLC processor or an internal string. Once you set up and enable this operation it is performed transparently in the background while the BASIC module executes a BASIC program in the foreground.

You can transfer the data low byte first, then high byte or high byte first, then low byte to the BASIC module port. You can also store the data in a string for the BASIC program to access. The byte swap selection (low byte first, then high byte, or high byte first, then low byte) of the last CALL 34 or CALL 33 you executed determines the data packing method for all ports CALL 34 enables.

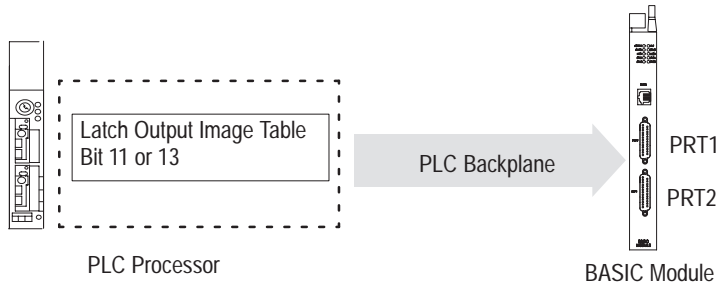
The low byte of the first word of the source file contains the character count (byte count) of the data you want to transfer. If the byte count is larger than the file, you can only transfer the maximum number of bytes within the file. You do not use the high byte of the first word.

Execute CALL 34 to set up the data transfer parameters. After you execute the call, the BASIC module gets data from the BTW buffer and transfers it to port PRT1, port PRT2, or an internal string.

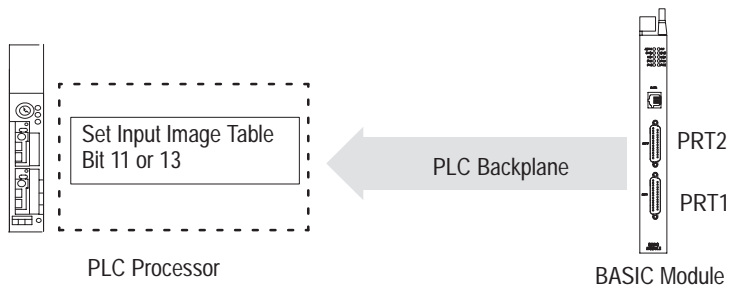
1. The ladder logic program of the PLC processor builds the data buffer. The ladder logic then determines the byte count of the file you want to transfer and places it into the lower byte of the BTW buffer word 1 to be transferred. This word plus the data comprise the data file to be transferred.



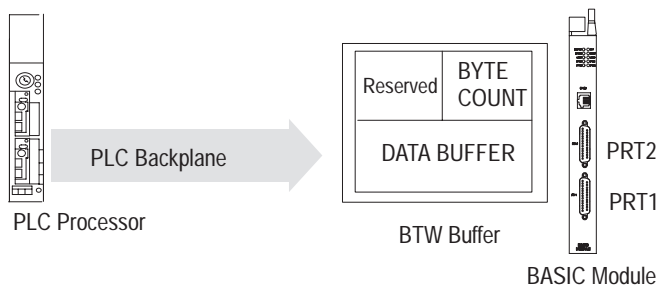
- The ladder logic program of the PLC processor must latch the output image table, bit 11 or bit 13 to inform the BASIC module that valid data is available. Bit 11 indicates that data is available for PRT1 and bit 13 indicates that data is available for PRT2.



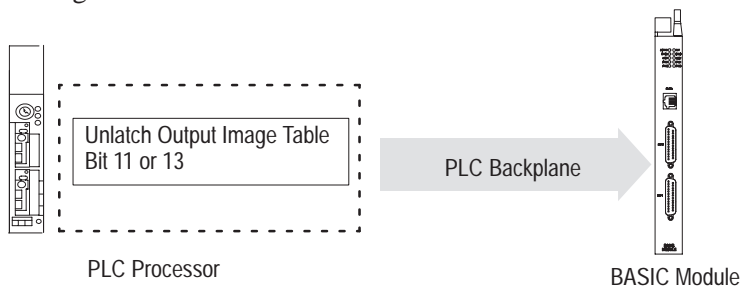
- The BASIC module sets the input image table bit 11 or bit 13 to inform the PLC processor that the BASIC module is ready for the transfer.



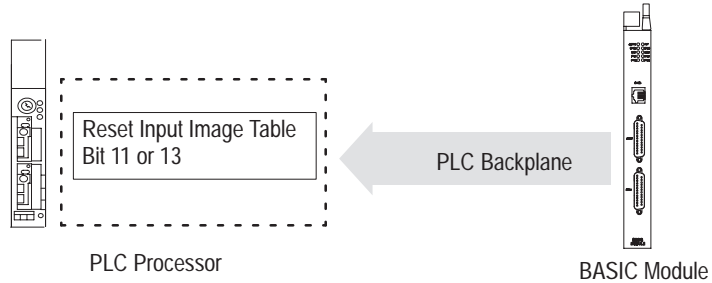
- The BASIC module automatically transfers the data to the destination serial port (PRT1 or PRT2) from the BTW buffer and performs the block transfer.



- The ladder logic program of the PLC processor unlatches output image table bit 11 or bit 13.



- The BASIC module resets the input image table bit 11 or bit 13 on the same end of scan cycle in which the output image table bit 11 or bit 13 was reset.



Transfers continue in this manner until you re-execute the call for the port with different input parameters. If this occurs, the previous CALL 34 for the port is automatically disabled and the new CALL 34 takes effect. You cannot execute multiple CALL 34s for the same port in parallel. However, you can activate port PRT1 and port PRT2 simultaneously by issuing separate CALL 34s for these ports.

Input and Output Arguments

This CALL has five input arguments and one output argument.

Argument	Description	Page
input 1	destination of the data	12 -31
input 2	always 0	12 -32
input 3	always 1	12 -32
input 4	internal string number	12 -32
input 5	byte swap selection	12 -32
output 1	call status	12 -32

Input Argument One

The first input argument is the destination of the data. You can choose the port number (1 or 2) and/or the internal string:

- 0 = disable CALL 34 for all active ports and strings enabled by earlier CALL 34s
- 1 = internal string only
- 2 = port PRT1
- 3 = internal string and port PRT1
- 4 = port PRT2
- 5 = internal string and port PRT2

If you choose an internal string (1, 3, or 5), the first character of the string contains the byte count. The second character (transaction number) increments to inform the BASIC module that new data is in the string. The value of this character wraps around from 255 to 0. The data from the source buffer begins with the third character of the string.

Input Argument Two

The second input argument is always 0.

Input Argument Three

The third input argument is always 1.

Input Argument Four

The fourth input argument is the internal string number. If the second input argument does not select internal string usage, the value of this input argument is ignored (but you must still PUSH it). If the data exceeds the string length, the remaining data is truncated.

Input Argument Five

The fifth input argument is the byte swap selection:

- 0 = data bytes transferred from the BTW buffer are *not* swapped when passed to the BASIC module port or string. The data transfer order is low byte first, then high byte per word. The low byte of the first word in the source buffer contains the byte count.
- 1 = data bytes transferred from the BTW buffer are swapped when passed to the BASIC module port or string. The data transfer order is high byte first, then low byte per word. Swapping does not affect the first word. The low byte of the first word still contains the byte count.

The last CALL 34 you executed determines the byte swap option for all active CALL 34 and CALL 33 commands you previously executed.

Output Argument One

The output argument is the status of the call. It has these values:

- 0 = successful
- 1 = disabled
- 2 = bad input parameter
- 3 = PRT2 is chosen but it is enabled for DF1 protocol—call not executed
- 4 = string is too small
- 5 = string is not dimensioned
- 6 = JW5 not in 16-point position

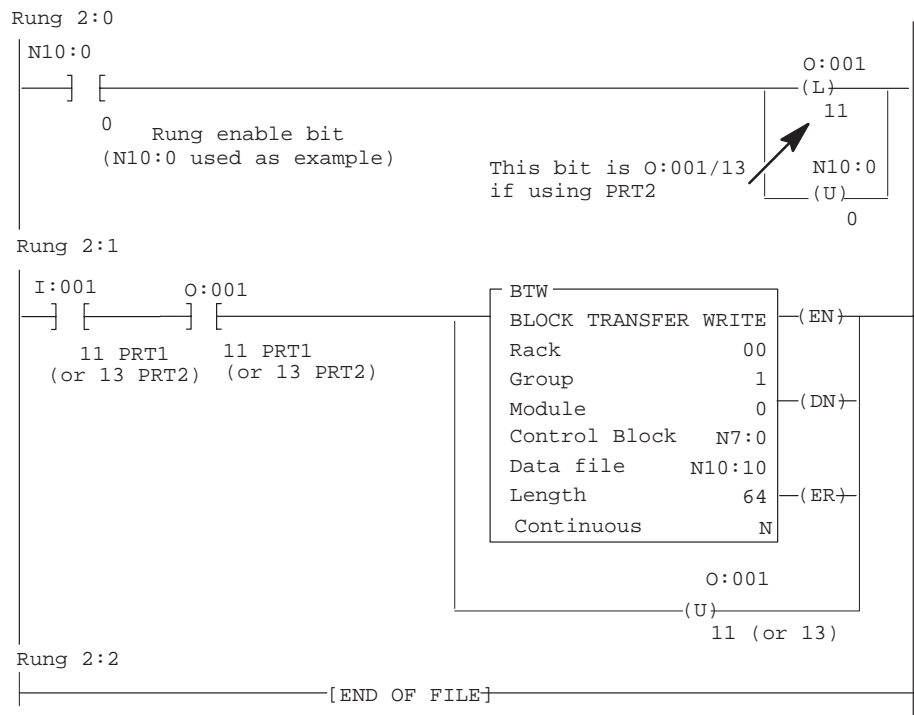
Syntax

PUSH destination port number and/or internal string
PUSH 0
PUSH 1
PUSH string number
PUSH byte swap selection
CALL 34
POP CALL 34 status

Example

```
>01 PUSH 64 : CALL 4 : REM SET BLOCK TRANSFER WRITE
    LENGTH
>10 REM ENABLE CALL 34 INTERRUPTS
>20 PUSH 2 : REM SEND DATA TO PRT1
>30 PUSH 0 : REM ALWAYS 0
>40 PUSH 1 : REM ALWAYS 1
>50 PUSH 0 : REM STRING NUMBER/NOT USED HERE
>60 PUSH 1 : REM ENABLE BYTE SWAPPING
>70 CALL 34
>80 POP S : REM STATUS OF CALL SETUP
>90 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 34 SETUP"
```

Sample Ladder Logic



CALL 35: Retrieve Numeric Input Character from PRT2 Port

Use this routine to retrieve the current character in the 255 character, PRT2 port receive buffer and convert it to its decimal representation. The PRT2 port receives data your device transmits and stores it in this buffer.

You can use the GET# statement (page 11 -12) in place of CALL 35.

Input and Output Arguments

This routine has no input arguments and one output argument. If there is not a character in the PRT2 receive buffer, the output argument is 0 (null). If there is a character, the output argument is the ASCII value of that character.

Important: A 0 (null) is a valid character in some communication protocols. Use CALL 36 to determine the actual number of characters in the buffer.

Syntax

```
CALL 35  
POP ASCII value of character
```

Example

```
>10 CALL 35  
>20 POP X  
>30 IF X = 0 THEN GOTO 10  
>40 PRINT CHR(X)
```

This program shows how to read a number from the PRT2 port. A device connected to the PRT2 port sends the number, followed by the character "r" (e.g. 51r). The "r" indicates the end of the number. The BASIC module unpacks the characters and combines them to form the number 51 and the character "r".

```
>10 REM THIS PROGRAM DEMONSTRATES HOW TO READ A NUMBER FROM THE  
>20 REM PRT2 PORT. A DEVICE CONNECTED TO THE PRT2 PORT  
>30 REM SENDS THE NUMBER AS A SERIES OF ASCII CHARACTERS,  
>40 REM FOLLOWED BY THE CHARACTER R. R INDICATES END OF THE NUMBER.  
>50 T=0 : REM T WILL HOLD THE NUMBER RECEIVED FROM THE REMOTE DEVICE.  
>60 CALL 35 : POP X : REM GET A CHARACTER FROM THE PRT2 PORT  
>70 REM IF THE CHARACTER IS A VALID DIGIT, ADD IT TO T  
>80 IF (X>=48).AND.(X<=57)T=(T*10)+(X-48)  
>90 IF X=114 GOTO 110 : REM CHARACTER 114 IS A LOWER CASE R  
>100 GOTO 60  
>110 PRINT "THE NUMBER IS", T
```


CALL 36: Get the Number of Characters in the PRT2 Port Buffer

Use this routine to retrieve the number of characters in the buffer you choose.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the buffer you want to examine:

- 0 = transmit buffer
- 1 = receive buffer

The output argument is the number of characters in the specified buffer.

Syntax

```
PUSH buffer to examine  
CALL 36  
POP number of characters
```

Example

```
>10 PUSH 0: REM EXAMINES THE TRANSMIT BUFFER  
>20 CALL 36  
>30 POP X: REM GET THE NUMBER OF CHARACTERS  
>40 PRINT "NUMBER OF CHARACTERS IN OUTPUT BUFFER IS", X  
>50 END
```

CALL 37: Clear the PRT2 Port Buffers

Use this routine to clear the PRT2 port receive and/or transmit buffer.

Input and Output Arguments

This routine has one input argument and no output argument. The input argument is the buffer you want to clear:

- 0 = transmit buffer
- 1 = receive buffer
- 2 = both buffers

Syntax

```
PUSH buffer to clear  
CALL 37
```

Example

```
>10 PUSH 0: REM CLEAR THE TRANSMIT BUFFER  
>20 CALL 37
```

CALL 38: Expanded ONERR Restart

Use CALL 38 to expand the type of errors the ONERR statement (page 11 -23) traps and handles.

The ONERR statement only allows the BASIC module to jump to an error handling routine when it encounters an arithmetic error (number too large, number too small, bad argument or division by zero occurs). All other errors cause the module to enter Command mode. When you initiate CALL 38, you allow ONERR to service other errors (except hardware errors—watchdog, time-out, RAM failure, etc.) the BASIC module encounters instead of returning to Command mode.

If any error occurs that causes a restart, stacks are cleared. Variables and ports, however, are not re-initialized. This call has no effect until you execute the ONERR statement within the program. This call is reset when the BASIC module returns to the Command mode. You must re-execute CALL 38 every time you enter Run mode.

Tip

Use CALL 0 (page 12 -2) if you want to reset the module within the error routine.

Input and Output Arguments

This routine has one input and no output arguments. The input argument determines if the expanded ONERR function is enabled or disabled:

- 0 = disable the expanded ONERR restart
- 1 (or any other number) = enable the expanded restart

If you perform an XBY in the error routine, this a list of the status codes you might receive.

Status code	Description
01	BASIC module attempted to call an illegal call number
02	port has been assigned an invalid parameter
03	string has not been dimensioned
04	defined string length is too small for operation
05	memory has not been allocated for this string
06	attempted to transfer to a RAM or ROM program that did not exist
07	command or call can only be executed from Command mode
08	user PROM has invalid checksum
09	this statement or call requires a user PROM; no user PROM is installed
10	divide by zero
11	DH-485 call executed and DH485 port not enabled
12	argument stack problem
13	syntax error
14	control stack problem
15	array size problem
16	internal processor stack problem

Status code	Description
17	no DATA available for READ
18	DF1 cannot be enabled (JW4 in wrong position)
19	<ul style="list-style-type: none"> • illegal use of PRT2 while DF1 is enabled • illegal use of PRT2 while background DF1 task is enabled • attempted to transmit DF1 packet before DF1 is enabled • attempted to transmit DF1 packet of incorrect length
20	arithmetic overflow (value too large for range)
21	bad line number
22	JW5 in 8-point position
30	arithmetic underflow (value too small for range)
40	bad argument

Syntax

```
PUSH 0 or 1
CALL 38
```

Example

```
>10 ONERR 160
>20 PUSH 1
>30 CALL 38
>40 PUSH 1000:REM PUSH NUMBER FOR CALL 20 CONVERSION +/-999
>50 PUSH 3:REM WORD 3 IN BTR BUFFER
>60 CALL 20:REM CONVERT TO 3-DIGIT SIGNED BCD
>70 X=100
>80 Y=0
>90 Z=X/Y
>100 END
>160 REM EXPANDED ONERR ROUTINE
>170 PRINT "ERROR CODE WAS",XBY(257)
>180 PRINT "AT LINE ",(256*XBY(69FDH)+XBY(69FEH))
>END
```

CALL 39 : 3.3-Digit Signed, BCD to BASIC Floating Point



Use this routine to convert 3.3-digit BCD from the PLC processor to BASIC floating point. See also CALL 26.

See Chapter 8 for more information.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the number (1-64) of the first word (3.3-digit BCD is sent to the BASIC module in two processor words) of the write-block-transfer buffer you want to convert from 3.3-digit, signed, fixed decimal BCD to BASIC floating point. The maximum values allowed are ± 999999 . The output argument is the value converted into BASIC floating point format. The sign bit is bit number 16.

Syntax

```
PUSH number of word (1 – 64) to be converted  
CALL 39  
POP converted value
```

Example

```
> 10 PUSH X  
> 20 CALL 39  
> 30 POP Y
```

CALL 40: Set the Wall Clock Time (Hour, Minute, Second)

Use this routine to set the wall clock time functions.

Important: The Series B, BASIC module does not update the wall clock for Daylight Savings Time. You must do this manually. (The Series A, BASIC module, revisions A, B, C and D update the fall time change correctly but update the spring time change on the third weekend of April instead of the first weekend.)

Input and Output Arguments

This routine has three input arguments and no output arguments. The input arguments are the wall clock time functions:

- H = hours (0 to 23)
- M = minutes (0 to 59)
- S = seconds (0 to 59)

Syntax

```
PUSH hours 0-23
PUSH minutes 0-59
PUSH seconds 0-59
CALL 40
```

Example

Program the wall clock for 1:35 pm (13:35 on a 24 hour clock).

```
>10 H=13: M=35: S=00 :REM HOURS=13; MINUTES=35; SECONDS=00
>20 PUSH H,M,S :REM PUSH HOURS, MINUTES, SECOND
>30 CALL 40 :REM CALL THE ROUTINE TO SET THE WALL CLOCK TIME
```

CALL 41: Set Wall Clock Date (Day, Month, Year)

Use this routine to set the wall clock date functions.

Input and Output Arguments

This routine has three input arguments and no output arguments. The input arguments are the wall clock date functions:

- D = day
- M = month
- Y = year

Syntax

```
PUSH date 1-31
PUSH month 1-12
PUSH year 0-99
CALL 41
```

Example

Program the wall clock for the 16th day of June 1994.

```
>10 D=16: M=06: Y=94 :REM DAY OF MONTH=16, MONTH=6, YEAR=94
>20 PUSH D,M,Y :REM PUSH DAY OF MONTH, MONTH, YEAR
>30 CALL 41 :REM CALL THE ROUTINE TO SET THE WALL CLOCK DATE
```

CALL 42: Set Wall Clock Day of Week

Use CALL 42 to set the day of the week.

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the day of the week:

- 1 = Sunday
- 2 = Monday
- 3 = Tuesday
- 4 = Wednesday
- 5 = Thursday
- 6 = Friday
- 7 = Saturday

Syntax

```
PUSH day of week 1-7
CALL 42
```

Example

```
>10 PUSH 3:REM DAY OF WEEK.
>20 CALL 42:REM DAY IS TUESDAY.
```

CALL 43: Retrieve Date/Time String

Use CALL 43 to retrieve the current date and time as a string.

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the number of the string to receive the date/time (dd/mm/yy and hh:mm:ss). You must use the STRING statement (see page 11 -37) to allocate a minimum of 18 characters for the string. This requires you to set the maximum length for all strings to at least 18 characters.

Syntax

```
PUSH number of string to receive date/time  
CALL 43
```

Example

```
>10 STRING 100,18  
>20 PUSH 1: CALL 43: REM PUT DATE/TIME IN STRING 1  
>30 PRINT $(1)  
>40 END
```

CALL 44: Retrieve Date Numeric (Day, Month, Year)

Use CALL 44 to retrieve the current date on the argument stack as three numbers.

Input and Output Arguments

This routine has no input arguments and three output arguments. The output arguments are the day, month and year in that order.

Syntax

```
CALL 44  
POP day  
POP month  
POP year
```

Example

```
>10 REM DATE RETRIEVE - NUMERIC EXAMPLE  
>20 CALL 44 : REM INVOKE THE UTILITY ROUTINE  
>30 POP D,M1,Y: REM GET THE DATA FROM THE 30 STACK  
>40 PRINT "CURRENT DATE IS", Y,M1,D  
>50 END  
>RUN  
CURRENT DATE IS 84 12 25
```

CALL 45: Retrieve Time String

Use CALL 45 to retrieve the current time in a string (hh:mm:ss).

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the number of the string to receive the time. You must use the STRING statement (see page 11 -37) to allocate a minimum of 8 characters for the string.

Syntax

```
PUSH number of string to receive the time  
CALL 45
```

Example

```
>10 STRING 100,8  
>20 PUSH 1: CALL 45: REM put time in string 1  
>30 PRINT $(1)  
>40 END  
>READY  
>RUN  
15: 45: 27
```

CALL 46: Retrieve Time Numeric

Use CALL 46 to retrieve the time of day in numeric form.

Input and Output Arguments

This routine has no input arguments and three output arguments. The output arguments are hours, minutes and seconds in that order.

Syntax

```
CALL 46  
POP hour  
POP minute  
POP second
```

Example

```
>10 REM TIME IN VARIABLES EXAMPLE  
>20 CALL 46 : REM GET WALL CLOCK TIME  
>30 POP H,M,S  
>40 PRINT "CURRENT TIME IS", H,M,S  
>50 END  
>RUN  
CURRENT TIME IS 13 44 54  
READY  
>
```


CALL 47: Retrieve Day of Week String

Use CALL 47 to retrieve the current day of week as a three character string.

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the number of the string to receive the day of week. You must use the STRING statement (see page 11 -37) to allocate a minimum of 3 characters/string. Strings returned are SUN, MON, TUE, WED, THU, FRI, SAT.

Syntax

PUSH number of string to receive the day of the week
CALL 47

Example

```
>10 STRING 100,3
>20 PUSH 0 : CALL 47
>30 PRINT "Today is ",$(0)
>RUN
Today is THU
```

CALL 48: Retrieve Day of Week Numeric

Use CALL 48 to retrieve the current day of week on the argument stack as a number.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument is the day of the week as a number:

- 1 = Sunday
- 2 = Monday
- 3 = Tuesday
- 4 = Wednesday
- 5 = Thursday
- 6 = Friday
- 7 = Saturday

Syntax

CALL 48
POP *day of the week (1-7)*

Example

```
>10 REM DAY OF WEEK RETRIEVE - NUMERIC EX
>20 CALL 48: REM INVOKE UTILITY TO GET D.O.W.
>30 POP D
```

CALL 49: Read Remote DH-485 SLC Data File



Tip



Important: This call requires the BASIC module to be in 16 point mode (page 1 -7).

Use CALL 49 to read up to 63 words of data from a remote DH-485 node and place in the BTR buffer or a string within the BASIC module.

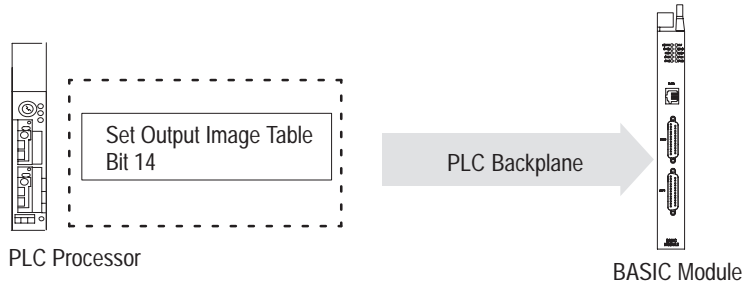
This call is useful for reading bar code data and sending it to the PLC processor or an internal string. Once you set up and enable this operation, it is performed transparently in the background while the BASIC module executes a BASIC program in the foreground.

Refer to the DF1 Protocol and Command Set Reference Manual (publication number 1770-6.5.16) for detailed information on DH-485.

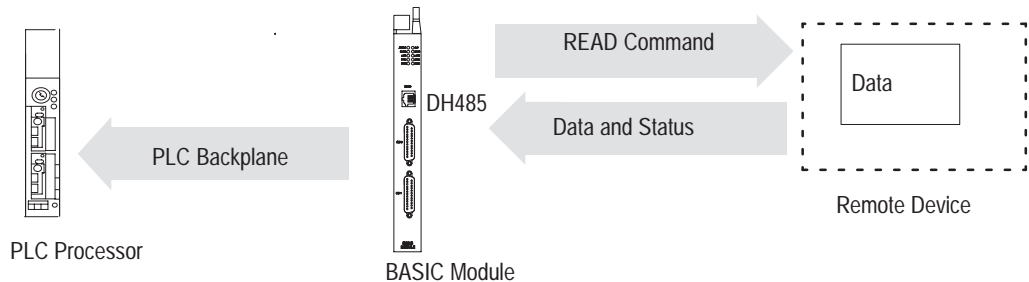
If you choose an internal string, the first character increments to inform the BASIC module that new data is in the string. The value of this character wraps around from 255 to 0.

Execute CALL 49 once to set up the data transfer parameter. PLC handshaking is used to initiate and notify completion of the transfer. The BASIC module sends the DH-485 READ command you configure in in this call to the remote DH-485 device you designate on the network.

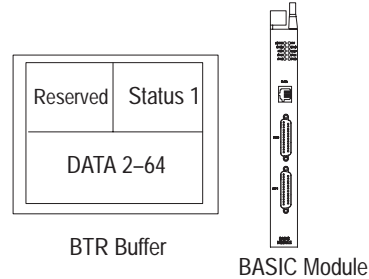
1. The local PLC processor sets the output image table bit 14 to inform the BASIC module to execute the DH-485 READ command you configured in CALL 49.



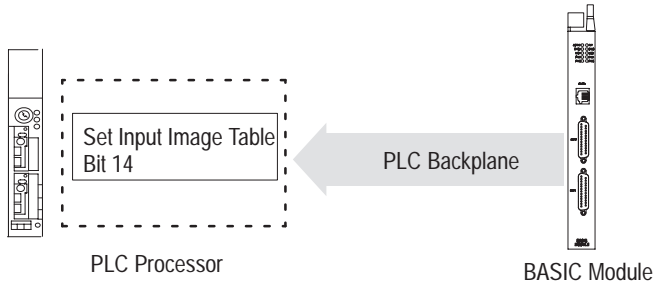
2. The BASIC module automatically issues the appropriate DH-485 READ command to the remote device on the DH-485 network. The data and status are sent back to the BASIC module.



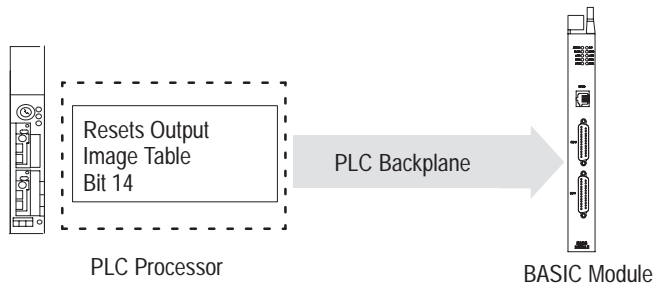
- When data is available, the BASIC module transfers the data into BTR buffer. The DH-485 status word is placed in lower byte of word 1. The upper byte of BTR word 1 is reserved.



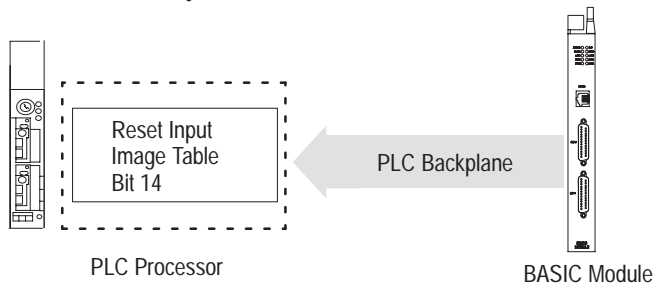
- The BASIC module sets the input image table, bit 14 and performs a block transfer read.



- The local PLC receives the data and status from the block transfer and then resets output image table bit 14 to inform the BASIC module that it received data.



- The BASIC module resets the input image table bit 14 on the same end of scan cycle in which the block transfer was completed.



This call is active until you re-execute it with different input parameters.

Input and Output Arguments

This call has ten input arguments and one output argument.

Argument	Description	Page
input 1	type of DH-485 READ	12 -46
input 2	node address of the DH-485 remote device (0 through 31)	12 -46
input 3	file number on the DH-485 remote device (0 through 255)	12 -46
input 4	file type to be read from the remote device	12 -47
input 5	starting word offset within the file on the remote device (0 through 32766)	12 -47
input 6	number of words to be transferred	12 -47
input 7	the message time-out value	12 -47
input 8	the selection of the destination file and/or string	12 -48
input 9	always 1	12 -48
input 10	the string number	12 -48
output 1	call status	12 -48

To disable this call, you must PUSH a 0 into the first input parameter. All other parameters are ignored, but you must still PUSH them.

Input Argument One

The first input argument is the type of DH-485 READ command issued:

- 0 = disable the previously executed CALL 49
- 1 = common interface file (CIF) read
- 2 = SLC typed read

Input Argument Two

The second input argument is the node address of the DH-485 remote device (0 through 31). If the number is not within this range, the status equals 2 and the read message does not occur.

Input Argument Three

The third input argument is the file number on the DH-485 remote device (0 through 255). If the number is not within this range, the status equals 2 and the read message does not occur. The parameter is ignored if you choose the Common Interface File (CIF) in the first parameter. The CIF is always file 9.

Input Argument Four

The fourth input argument is the file type to be read from the remote device. This number is ignored if the CIF is chosen in the first parameter (assumes integer file). If the file type is not one of these listed below, the status equals 2 and the read message does not take place. Enter the file type code as shown below when you PUSH the fourth input parameter.

File type	File type code	Words/Element
integer file	ASC(N)	1 word/element
counter file	ASC(C)	3 words/element
timer file	ASC(T)	3 words/element
bit file	ASC(B)	1 word/element
control file	ASC(R)	3 words/element

Input Argument Five

The fifth input argument is the starting word offset within the file on the remote device (0 through 32766). If the number is not within this range, the status equals 2 and the transfer does not occur. (The SLC 500 processor only supports 0 through 255 words per file.)

Input Argument Six

The sixth input argument is the number of elements you want to transfer. If the number is not within the range shown, the status equals 2 and the transfer does not occur. SLC 5/01 and SLC 5/02 processors support transfers up to 41 words maximum.

File type code	Valid length range
ASC(N)	1 to 63
ASC(C)	1 to 21
ASC(T)	1 to 21
ASC(B)	1 to 63
ASC(R)	1 to 21
common interface file	1 to 63

Important: When selecting the BTR buffer as the destination in Input Argument Eight, the valid length range in Input Argument Six assumes that the proper block-transfer length has been set.

Input Argument Seven

The seventh input argument is the message time-out value. This value (1 through 255) corresponds to the number of hundreds of milliseconds that are allowed to receive the read response (0.1 through 25.5 seconds). If the read response is not received within this time, the message aborts with the status equal to 55 in the BTR word 1. If the time-out value is not within the range (1 through 255), the output status equals 2 and the transfer does not take place.

Input Argument Eight

The eighth input argument is the selection of the BTR buffer and/or string:

- 0 = BTR buffer
- 2 = internal string
- 4 = BTR buffer and internal string

If you choose internal string (2), you can execute CALL 29 (page 12 -18) to initiate each data transfer without requiring PLC processor interaction.

Input Argument Nine

The ninth input argument is always 1.

Input Argument Ten

The tenth input argument is the string number. If the eighth input argument does not select internal string usage, the value of this input argument is ignored, but you must still PUSH it.

Output Argument One

The output argument is the status of the call. It has these values:

- 0 = successful
- 1 = disabled
- 2 = bad input parameter
- 3 = port DH-485 not enabled (DF1 enabled)
- 4 = string is too small
- 5 = string is not dimensioned
- 6 = JW5 not in 16-point position

Whenever an attempt is made to read a remote packet, the status of the read is placed into BTR word 1. These values have the same definition as the output values in CALL 92 (page 13 -26). The status becomes valid when the BASIC module sets the input image table bit 14.

Syntax

```
PUSH type of DH-485 READ command
PUSH remote node address
PUSH remote file number
PUSH remote file type
PUSH starting word offset of remote file
PUSH number of words to be transferred
PUSH message time-out value
PUSH selection of destination file (BTR buffer)
PUSH 1
PUSH string number
CALL 49
POP CALL 49 status
```

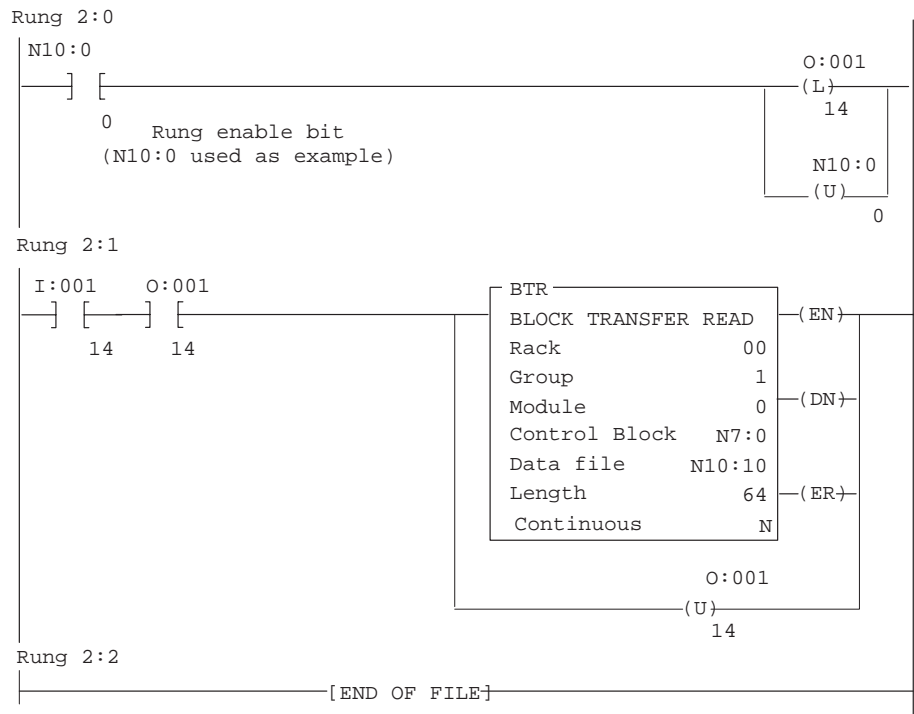
Example

```

>10 REM ENABLE REMOTE DH-485 READ COMMAND INTERRUPT
>15 PUSH 64: CALL 4: REM SET BLOCK TRANSFER WRITE LENGTH
>16 PUSH 64: CALL 5: REM SET BLOCK TRANSFER READ LENGTH
>20 PUSH 2 : REM SLC TYPED READ COMMAND
>30 PUSH 2 : REM NODE ADDRESS OF REMOTE SLC
>40 PUSH 7 : REM FILE NUMBER OF REMOTE SLC
>50 PUSH ASC(N) : REM FILE TYPE OF REMOTE SLC
>60 PUSH 100 :REM REMOTE ELEMENT OFFSET INTO REMOTE SLC FILE
>70 PUSH 20 : REM NUMBER OF ELEMENTS TO BE TRANSFERRED
>80 PUSH 5 : REM MESSAGE TIMEOUT (X100MS)
>90 PUSH 0 : REM DESTINATION FILE TO PUT DATA (BTR BUFFER)
>100 PUSH 1 : REM ALWAYS 1
>110 PUSH 0 :REM STRING NUMBER-NOT AVAILABLE FOR THIS EXAMPLE
>120 CALL 49
>130 POP S
>140 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 49 SETUP"

```

Sample Ladder Logic



CALL 50: Write to Remote DH-485 SLC Data



Tip



Important: This call requires the BASIC module jumper JW5 to be in 16 point mode (page 1 -7).

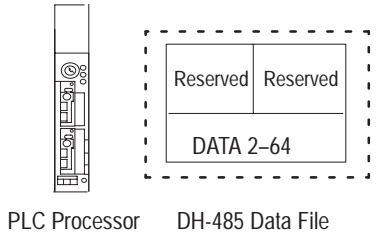
Use CALL 50 to write up to 63 words of data from the BTW buffer and/or a string within the BASIC module to the remote DH-485 file at the node address you designate.

This call is useful for writing to a display device directly from the PLC processor or an internal string. Once you set up and enable this operation, it is performed transparently in the background while the BASIC module executes a BASIC program in the foreground.

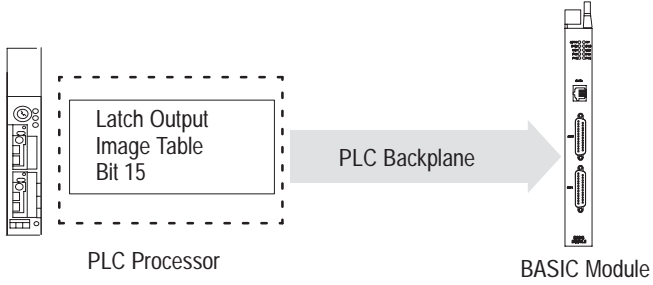
Refer to the DF1 Protocol and Command Set Reference Manual (publication number 1770-6.5.16) for detailed information on DH-485.

If you choose an internal string, the first character (transaction number) increments on successful completion of the write to inform the BASIC module that data was sent. The value of the transaction number wraps around from 255 to 0. Execute CALL 50 once to set up data transfer parameters.

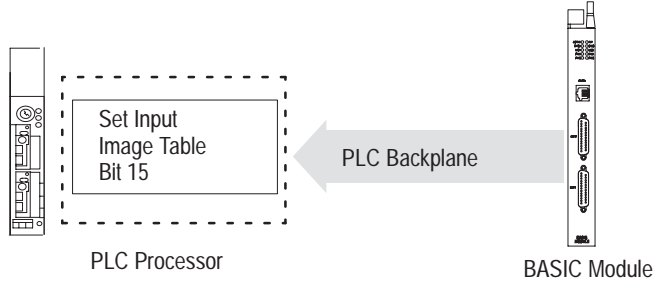
1. Local PLC processor ladder logic builds a file with the DH-485 data.



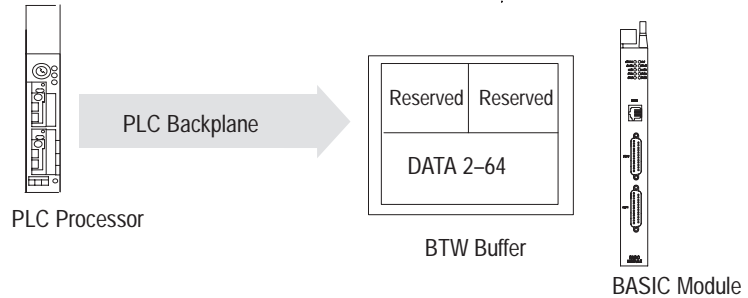
2. The PLC processor latches output image table, bit 15 to inform the BASIC module that data can be transferred.



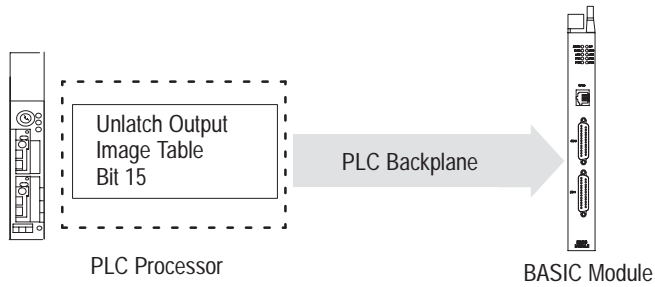
- The BASIC module sets bit 15 in the input image table to inform the PLC processor that a block transfer write will be performed.



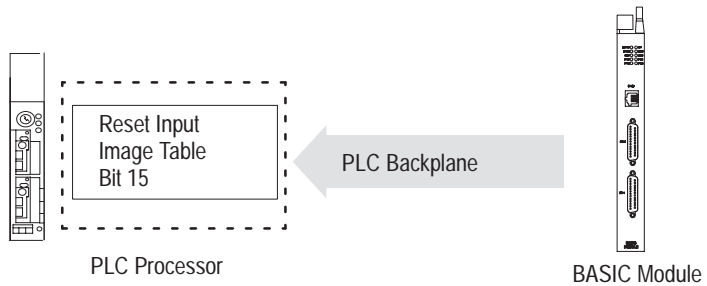
- The BASIC module performs a block transfer to receive the data.



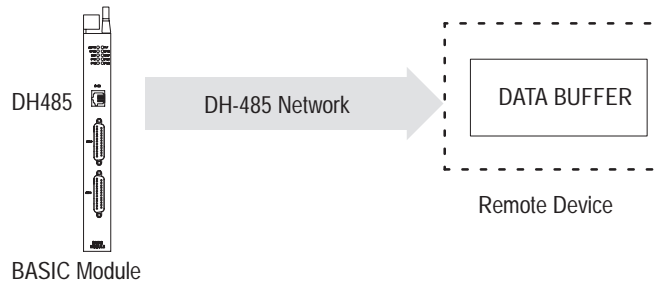
- The PLC processor unlatches bit 15 in the output image table.



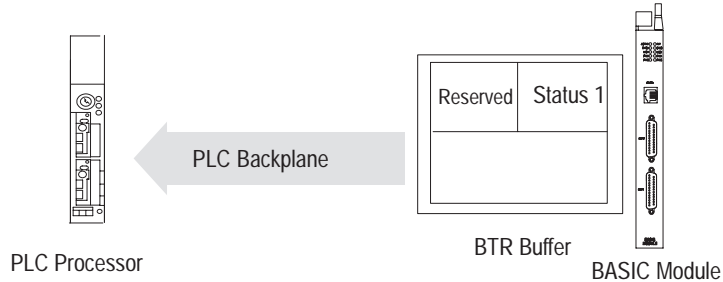
- The BASIC module resets bit 15 in the input image table.



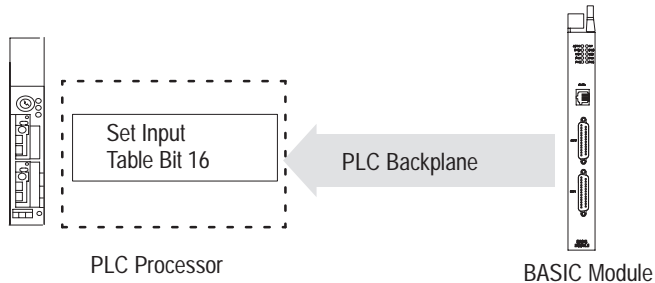
7. The BASIC module assembles the DH-485 packet and sends it to the remote device on the DH-485 network



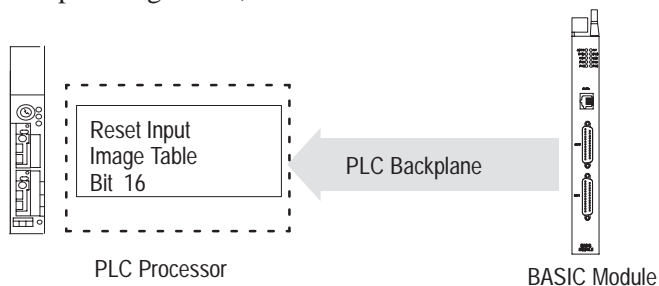
8. The BASIC module places the DH-485 transfer status into the BTR buffer word 1.



9. The BASIC module sets the input image table, bit 16 to inform the local PLC processor that valid data is transferred and the status of the transfer is available. The PLC performs a block transfer to retrieve the status.



10. The BASIC module detects a successful block transfer and resets the input image table, bit 16.



Once this call is active, it remains active and sends data to the remote node whenever the PLC processor handshaking is accomplished.

Input and Output Arguments

This call has ten input arguments and one output argument.

Argument	Description	Page
input 1	type of DH-485 WRITE	12 -53
input 2	node address of the DH-485 remote device (0 through 31)	12 -53
input 3	file number on the DH-485 remote device (0 through 255)	12 -53
input 4	file type to be written to the remote device	12 -54
input 5	starting word offset within the file on the remote device (0 through 32766)	12 -54
input 6	number of words to be transferred	12 -54
input 7	the message time-out value	12 -54
input 8	the selection of the source BTW buffer or internal string	12 -55
input 9	always 1	12 -55
input 10	the string number	12 -55
output 1	call status	12 -55

To disable this call, you must PUSH a zero into the first input parameter. All other parameters are ignored but you must still PUSH them.

Input Argument One

The first input argument is the type of DH-485 WRITE command issued:

- 0 = disable the previously executed CALL 50
- 1 = common interface file write
- 2 = SLC typed write

Input Argument Two

The second input argument is the node address of the DH-485 remote device on the DH-485 network (0 through 31). If the number is not within this range, the status equals 2 and the write message does not occur.

Input Argument Three

The third input argument is the file number on the DH-485 remote device (0 through 255). If the number is not within this range, the status equals 2 and the write message does not occur.

Input Argument Four

The fourth input argument is the file type you want to write to the remote device. This number is ignored if you choose the CIF in the first parameter (assumes integer file). If the file type is not one of those listed, the status equals 2 and the write message does not take place. Enter the file type code as shown:

File type	File type code	Words/Element
integer file	ASC(N)	1 word/element
counter file	ASC(C)	3 words/element
timer file	ASC(T)	3 words/element
bit file	ASC(B)	1 word/element
control file	ASC(R)	3 words/element

Input Argument Five

The fifth input argument is the starting word offset within the file on the SLC remote device (0 through 32766). If the number is not within this range, the status equals 2 and the transfer does not occur.

Input Argument Six

The sixth input argument is the number of elements you want to transfer. If the number is not within the range shown, the status equals 2 and the transfer does not occur.

File type code	Valid length range
ASC(N)	1 to 40
ASC(C)	1 to 13
ASC(T)	1 to 13
ASC(B)	1 to 40
ASC(R)	1 to 13
common interface file	1 to 40

Input Argument Seven

The seventh input argument is the message time-out value. This value (1 through 255) corresponds to the number of hundreds of milliseconds that are allowed to receive the write response (0.1 through 25.5 seconds). If the write response is not received within this time, the message aborts and the status equals 55 in BTR word 1. If the time-out value is not within the range (1 through 255), the output argument status equals 2 and the transfer does not take place.

Input Argument Eight

The eighth input argument is the selection of the source BTW buffer or the internal string:

- 0 = BTW buffer
- 2 = internal string

If you choose internal string (2), you can execute CALL 29 (page 12 -18) to initiate each data transfer without requiring PLC processor interaction. The output table bit 15 also initiates a string transaction.

Input Argument Nine

The ninth input argument is always 1.

Input Argument Ten

The tenth input argument is the string number. If the eighth input argument does not select internal string usage, the value of this input argument is ignored, but you must still push it.

Output Argument One

The output argument is the validation of the call parameters. It has these values:

- 0 = successful
- 1 = disabled
- 2 = bad input parameter
- 3 = port DH-485 not enabled (DF1 enabled)
- 4 = string is too small
- 5 = string is not dimensioned
- 6 = JW5 is not in 16-point position

Whenever you attempt to write to a remote node, the BASIC module places the status of the write into the BTR buffer word 1. The status values have the same definition as the output values in CALL 93 (page 13 -29).

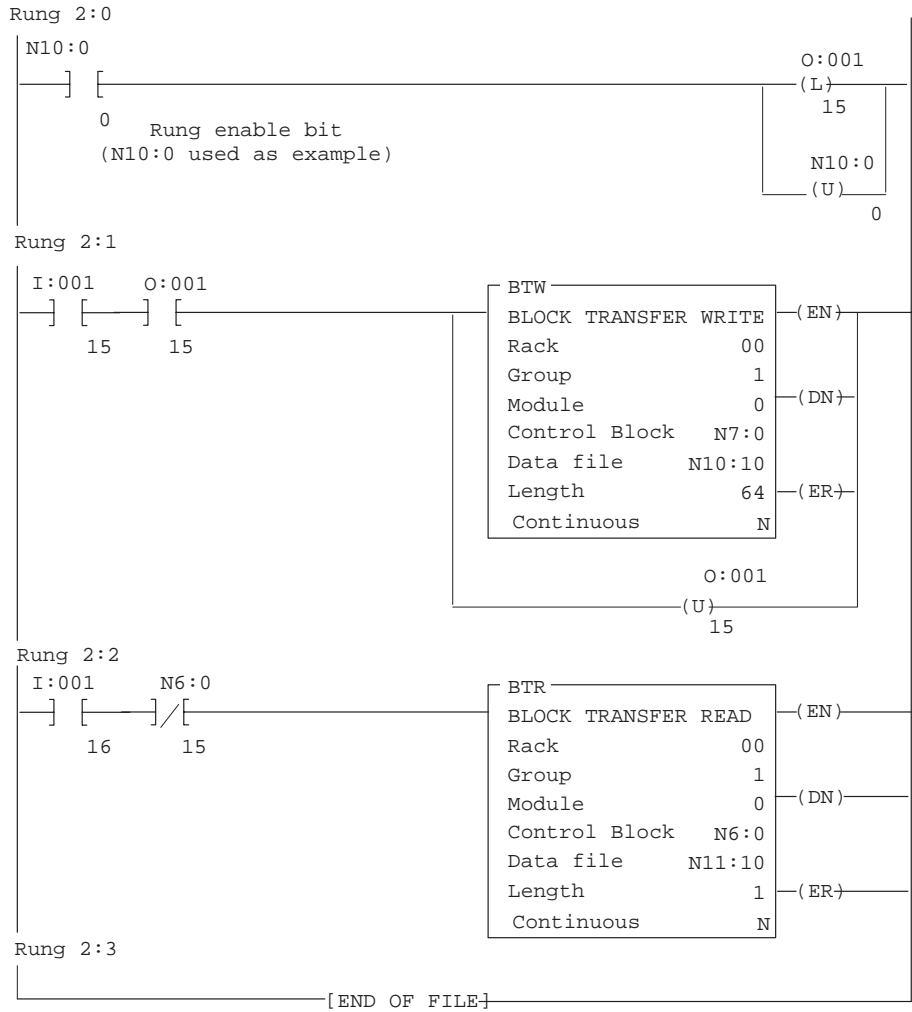
Syntax

PUSH type of DH-485 WRITE command
PUSH remote DH-485 device node number
PUSH remote DH-485 device file number
PUSH remote DH-485 device file type
PUSH element offset into destination file
PUSH number of elements to be transferred
PUSH message time-out value(X100MS)
PUSH block transfer write buffer
PUSH 1
PUSH Internal string number
CALL 50

Example

```
>10 PUSH 64 : CALL 4 : REM SET BLOCK TRANSFER WRITE LENGTH
>20 PUSH 2 : REM SLC TYPED WRITE
>30 PUSH 1 : REM REMOTE SLC NODE NUMBER
>40 PUSH 7 : REM REMOTE SLC FILE NUMBER
>50 PUSH ASC(N) : REM REMOTE SLC FILE TYPE
>60 PUSH 0 : REM ELEMENT OFFSET INTO DESTINATION FILE
>70 PUSH 20 : REM NUMBER OF ELEMENTS TO BE TRANSFERRED
>80 PUSH 10 : REM MESSAGE TIME-OUT VALUE(X100MS)
>90 PUSH 0 : REM LOCAL BTW BUFFER
>100 PUSH 1 : REM ALWAYS 1
>110 PUSH 0 : REM INTERNAL STRING NUMBER-UNUSED FOR THIS EXAMPLE
>120 CALL 50
>130 POP S
>140 IF (S=1) THEN PRINT "CALL 50 DISABLED"
>150 IF (S=2) THEN PRINT "CALL 50 BAD INPUT PARAMETER"
>160 IF (S=3) THEN PRINT "PORT DH485 NOT ENABLED"
```

Sample Ladder Logic



CALL 51

Undefined. If you execute an undefined call, you receive the error message, "ERROR-UNSUPPORTED CALL."

CALL 52: Retrieve Date String

Use CALL 52 to retrieve the current date in a string (dd/mm/yy).

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the number of the string to receive the date. You must use the STRING statement (see page 11 -37) to allocate a minimum of 9 characters for the string.

Syntax

```
PUSH number of string to receive the date  
CALL 52
```

Example

```
>10 STRING 100,9  
>20 PUSH 1: CALL 52: REM PUT DATE IN STRING 1  
>30 PRINT $(1)  
>40 END  
>RUN  
30-JAN-94
```

CALL 53 - 59

Undefined. If you execute an undefined call, you receive the error message, "ERROR-UNSUPPORTED CALL."

CALL 60: String Repeat

Use this routine to repeat a character and place it in a string. You can use the string repeat when designing output formats. You cannot repeat more characters than the string's maximum length.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input argument is the number of times you want to repeat the character. The second input argument is the number of the string containing the character you want to repeat.

Syntax

```
PUSH number of times to repeat character  
PUSH number of string containing character to be repeated  
CALL 60
```

Example

```
>20 STRING 1000,50  
>30 $(1)="*"  
>40 PUSH 40: REM THE NUMBER OF TIMES TO REPEAT CHARACTER  
>50 PUSH 1: REM WHICH STRING CONTAINS CHARACTER  
>60 CALL 60  
>70 PRINT $(1)  
>80 END  
>RUN  
*****
```

CALL 61: String Append (Concatenation)

Use this routine to append one string to the end of another string. If the resulting string is longer than the maximum string length, the append characters are lost. This is a string concatenation assignment. (Ex. $\$(1)=\$(1)+\$(2)$).

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input argument is the string number of the string you want appended. The second input argument is the string number of the base string.

Syntax

```
PUSH string number of string to be appended  
PUSH string number of the base string  
CALL 61
```

Example

```
>10 STRING 200,20  
>20 $(1)="How are "  
>30 $(2)= "you?"  
>40 PRINT "BEFORE:",  
>50 PRINT "$1=",$(1)," $2=",$(2)  
>60 PUSH 2 :REM STRING NUMBER OF STRING TO BE APPENDED  
>70 PUSH 1 :REM BASE STRING NUMBER  
>80 CALL 61 :REM INVOKE STRING CONCATENATION ROUTINE  
>90 PRINT "AFTER:",  
>100 PRINT "$1=",$(1)," $2=",$(2)  
>110 END  
  
>RUN
```

```
BEFORE:      $1=How are           $2=you?  
AFTER:      $1=How are you?      $2=you?
```

CALL 62: Number to String Conversion

Use this routine to convert a number or numeric variable into a string. You must use the `STRING` statement (see page 11 -37) to allocate a minimum of 14 characters for the string. If the exponent of the value you want to convert is 100 or greater, you must allocate 15 characters. Error checking traps string allocation of less than 14 characters only.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input argument is the value you want to convert. The second input argument is the number of the string to receive the value.

Syntax

```
PUSH value to be converted  
PUSH number of string to receive the value  
CALL 62
```

Example

```
>10 STRING 100,14  
>20 INPUT "ENTER A NUMBER TO CONVERT TO A STRING",N  
>30 PUSH N  
>40 PUSH 1: REM CONVERT NUMBER TO STRING1  
>50 CALL 62: REM DO THE CONVERSION  
>60 PRINT $(1)  
>70 END
```

CALL 63: String to Number Conversion

Use this routine to convert the first decimal number found in the string you specify to a number, and place this number on the argument stack. Valid numbers and associated characters are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., E, +, -. The comma is not a valid number character and terminates the conversion.

Input and Output Arguments

This routine has one input argument and two output arguments. The input argument is the number of the string you want to convert. The first output argument is the validity value. The second output argument is the converted value. If a string contains a number followed by an E followed by a letter or non-numeric character, it is assumed that no number was found since the letter is not a valid exponent (UAB701EA returns a zero in the first output argument indicating that no valid number was in the string). If the string does not contain a legal value, a zero is returned. A valid value is between 1 and 255.

Syntax

```
PUSH value to be converted
CALL 63
POP validity value
POP actual value
```

Example

```
>10 STRING 100,14
>20 INPUT "ENTER A STRING TO CONVERT", $(1)
>30 PUSH 1: REM CONVERT STRING 1
>40 CALL 63: REM DO THE CONVERSION
>50 POP V,N
>60 IF V<>0 THEN PRINT $(1)," ",N: GO TO 80
>70 PRINT "INVALID OR NO VALUE FOUND"
>80 END
```

CALL 64: Find a String in a String

Use this routine to find a string within a string. It locates the first occurrence (position) of this string. This routine is similar to the ANSI BASIC INSTR\$(findstr\$,str\$). (Example: L=INSTR\$(1),(2))

Input and Output Arguments

This routine has two input arguments and one output argument. The first input is the string you want to find. The second input is the string you want to search for a match. The output argument is the location of the matched string. If the number is not zero then a match was located at the position the value of the return argument indicates.

Syntax

```
PUSH string number of string to be found
PUSH base string number
CALL 64
POP return value
```

Example

```
>10  REM SAMPLE FIND STRING IN STRING ROUTINE
>20  STRING 1000,20
>30  $(1)="456"
>40  $(2)="12345678"
>50  PUSH 1  :REM STRING NUMBER OF STRING TO BE FOUND
>60  PUSH 2  :REM BASE STRING NUMBER
>70  CALL 64  :REM GET LOCATION OF FIRST CHARACTER
>80  POP L
>90  IF L=0 THEN PRINT "NOT FOUND"
>100 IF L>0 THEN PRINT "FOUND AT LOCATION ",L
>110 END

>RUN

FOUND AT LOCATION 4

READY
```

CALL 65: Replace a String in a String

Use this routine to replace a string within a string.

Input and Output Arguments

This routine has three input arguments and no output arguments. The first input argument is the string number of the new string to replace the old string. The second input argument is the string number of the old string to be replaced by the new string. The third input argument is the base string's string number.

Syntax

```
PUSH string number of the replacement string  
PUSH string number of the string to be replaced  
PUSH base string number  
CALL 65
```

Example

```
>10 REM SAMPLE OF REPLACE STRING IN STRING  
>20 STRING 1000,20  
>30 $(0)="RED-LINES"  
>40 $(1)="RED"  
>50 $(2)="BLUE"  
>60 PRINT "BEFORE: $0=",$(0)  
>70 PUSH 2 :REM STRING NUMBER OF STRING TO REPLACE WITH  
>80 PUSH 1 :REM STRING NUMBER OF STRING TO BE REPLACED  
>90 PUSH 0 :REM BASE STRING NUMBER  
>100 CALL 65 :REM REPLACE STRING IN STRING ROUTINE  
>110 PRINT "AFTER: $0=",$(0)  
>120 END  
>RUN  
BEFORE: $0=RED-LINES  
AFTER: $0=BLUE-LINES
```

CALL 66: Insert String in a String

Use this routine to insert a string within another string.

Input and Output Arguments

This routine has three input arguments and no output arguments. The first argument is the position at which to begin the insert. The second argument is the string number of the characters you want to insert into the base string. The third argument is the string number of the base string.

Syntax

```
PUSH position to begin insert
PUSH string number of characters to be inserted
PUSH base string number
CALL 66
```

Example

```
>10  REM SAMPLE ROUTINE TO INSERT A STRING IN A STRING
>20  STRING 500,15
>30  $(0)+"1234590"
>40  $(1)+"67890"
>50  PRINT "BEFORE: 0$=",$(0)
>60  PUSH 6  :REM POSITION TO START THE INSERT
>70  PUSH 1  :REM STRING NUMBER OF STRING TO INSERT
>80  PUSH 0  :REM BASE STRING NUMBER
>90  CALL 66 :REM INVOKE INSERT A STRING IN A STRING
>91  REM: REM ROUTINE
>100 PRINT "AFTER: 0$=",$(0)
>110 END

>RUN

BEFORE:  0$=1234590
AFTER:   0$=123456789090
```

CALL 67: Delete String from a String

Use this routine to delete a string from within another string.

Important: This routine deletes only the first occurrence of the string.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first argument is the base string number. The second is the string number of the string you want to delete from the base string.

Syntax

```
PUSH base string number
PUSH string number of string to be deleted from base string
CALL 67
```

Example

```
>10 REM ROUTINE TO DELETE A STRING IN A STRING
>20 STRING 200,14
>30 $(1)="123456789012"
>40 $(2)="12"
>50 PRINT "BEFORE: $1=",$(1)
>60 PUSH 1 :REM BASE STRING NUMBER
>70 PUSH 2 :REM STRING NUMBER OF STRING TO BE DELETED
>80 CALL 67: REM INVOKE STRING DELETE ROUTINE
>90 PRINT "AFTER: $1=",$(1)
>100 END
>RUN

BEFORE: $1=123456789012
AFTER: $1=3456789012
```


CALL 68: Determine Length of a String

Use this routine to determine the length of a string. To properly determine the length of a string you must terminate the string with a CR character. If you use the ASC string operator (page 9 -14) to fill the string, you must add a CR as the last character to terminate the string.

Input and Output Arguments

This routine has one input and one output argument. The input is the string number on which the routine acts. The output is the actual number of non-carriage return (CR) characters in this string.

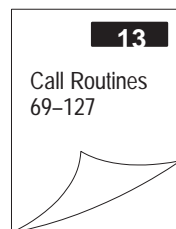
Syntax

```
PUSH number of string to determine length  
CALL 68  
POP number of characters in string
```

Example

```
>10 REM SAMPLE OF STRING LENGTH  
>20 STRING 100,10  
>30 $(1)="1234567"  
>40 PUSH 1 : REM BASE STRING  
>50 CALL 68 : REM INVOKE STRING LENGTH ROUTINE  
>60 POP L : REM GET LENGTH OF BASE STRING  
>70 PRINT "THE LENGTH OF",$(1)," IS ",L  
>80 END  
>RUN  
THE LENGTH OF 1234567 IS 7
```

What's Next?



Notes:

Call Routines 69–127

What's in This Chapter?



There are 128 BASIC calls. Calls 69 – 127 are described here. Calls 0 – 68 are described in Chapter 12. Chapter 7 gives you an overview of how to use these calls within your BASIC program. Use these calls within your BASIC program or from the command line.

Important: CALL numbers above 127 are not valid and cause the BASIC module error—ERROR CALL ARGUMENT OUT OF RANGE.

CALL	Description	Page
69	undefined	13 -2
70	ROM to RAM program transfer	13 -2
71	ROM/RAM to ROM program transfer	13 -3
72	RAM/ROM return	13 -4
73	battery-backed RAM disable	13 -5
74	battery-backed RAM enable	13 -5
75	undefined	13 -5
76	undefined	13 -5
77	protected variable storage	13 -6
78	set program port communication rate	13 -8
79	set active LED blinking state (no operation)	13 -8
80	check battery condition	13 -9
81	user PROM check and description	13 -10
82	check user memory module map	13 -11
83	display DH485 port setup	13 -11
84	transfer DH-485 CIF to BASIC input buffer	13 -12
85	transfer BASIC output buffer to DH-485 CIF file	13 -13
86	check DH-485 interface remote write status	13 -14
87	check DH-485 interface file remote read status	13 -15
88	BASIC floating point to PLC-5 floating point	13 -16
89	PLC-5 floating point to BASIC floating point	13 -17
90	read remote DH-485 data file to BASIC input buffer	13 -18
91	write BASIC output buffer to remote DH-485 data file	13 -22
92	read remote DH-485 CIF to BASIC input buffer	13 -26
93	write output buffer to remote DH-485 CIF file	13 -29
94	display current PRT1 port setup	13 -32
95	get number of characters in PRT1 buffers	13 -32
96	clear PRT1 receive/transmit buffers	13 -33
97	enable PRT2 DTR signal	13 -33
98	disable PRT2 DTR signal	13 -34

CALL	Description	Page
99	reset print head pointer	13 -34
100	download/program assembly language to EEPROM	13 -35
101	upload user (E)EPROM code to host	13 -35
102	undefined	13 -35
103	print PRT1 transmit buffer and pointer	13 -36
104	print PRT1 receive buffer and pointer	13 -37
105	reset PRT1 to default settings	13 -37
106	undefined	13 -38
107	undefined	13 -38
108	enable DF1 driver communications	13 -38
109	print the argument stack	13 -44
110	print the PRT2 port transmit buffer and pointer	13 -45
111	print the PRT2 receive buffer and pointer	13 -46
112	user LED control	13 -47
113	disable DF1 driver communications	13 -47
114	transmit DF1 packet	13 -48
115	check DF1 status	13 -49
116	call user defined assembly language routine	13 -50
117	get DF1 packet length	13 -51
118	PLC/SLC unsolicited writes	13 -52
119	reset PRT2 port to default settings	13 -56
120	clear BASIC module I/O buffers	13 -57
121	undefined	13 -57
122	read remote DF1 PLC data file	13 -58
123	write to remote DF1 PLC data file	13 -66
124	undefined	13 -74
125	undefined	13 -74
126	undefined	13 -74
127	undefined	13 -74

CALL 69

Undefined. If you execute an undefined call, you receive the error message, “ERROR-UNSUPPORTED CALL.”

CALL 70: ROM to RAM Program Transfer

Use this routine to shift program execution from a running ROM program to the beginning of the RAM program.

Important: The first line of the RAM program is not executed. We recommend that you make it a remark.

Important: There must be a next line in the ROM or RAM routine, otherwise unpredictable events could occur that may destroy the contents of RAM. For this reason always be sure that at least one END statement exists following a CALL 70 or 71.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 70
```

Example

```
>ROM5
>LIST
10 REM SAMPLE ROM PROG FOR CALL 70
20 PRINT "NOW EXECUTING ROM #5"
30 CALL 70 : REM GO EXECUTE RAM
40 END

>RAM
>LIST
10 REM SAMPLE RAM PROGRAM FOR CALL 70
20 PRINT "NOW EXECUTING RAM"
30 END

>ROM5
>RUN
NOW EXECUTING ROM #5
NOW EXECUTING RAM
```

CALL 71: ROM/RAM to ROM Program Transfer

Use this routine to transfer from a running ROM or RAM program to the beginning of any available ROM program.

Important: The first line of the ROM program is not executed. We recommend that you make it a remark.

Important: There must be a next line in the ROM or RAM routine, otherwise unpredictable events could occur that may destroy the contents of RAM. For this reason always be sure that at least one END statement exists following a CALL 70 or 71.

Input and Output Arguments

This routine has one input argument and no output arguments. The input is the ROM to which you want to transfer. If the ROM number does not exist an invalid program error displays and you enter the Command mode.

Syntax

```
PUSH beginning of ROM program
CALL 71
```

Example

```
>10 REM THIS ROUTINE CALLS AND EXECUTES A ROM ROUTINE
>20 INPUT "ENTER ROM ROUTINE TO EXECUTE",N
>30 PUSH N
>40 CALL 71
>50 END

>RUN

ENTER ROM ROUTINE TO EXECUTE 4
```

The user is now executing ROM 4 if it exists. If the ROM routine requested does not exist the result is:

```
PROGRAM NOT FOUND.
READY
>
```

CALL 72: RAM/ROM Return

Use this routine to return to the routine that called this ROM/RAM routine. Execution begins on the line after the line that called the ROM/RAM routine. This routine works one layer deep. You may go back to the last called program's next line.

Important: There must be a next line in the ROM or RAM routine, otherwise unpredictable events could occur that may destroy the contents of RAM. For this reason always be sure that at least one END statement exists following a CALL 70 or 71.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 72
```

Example

```
>ROM 1

>10  REM SAMPLE PROG FOR CALL 72
>20  PRINT "NOW EXECUTING ROM #1
>30  PUSH 3
>40  CALL 71 : REM EXECUTE ROM #3 THEN RETURN
>50  PRINT "EXECUTING ROM #1 AGAIN"
>60  END

>ROM 3

>10  REM THIS LINE WONT BE EXECUTED
>20  PRINT "NOW EXECUTING ROM #3"
>30  CALL 72
>40  END
```

With ROM #1 selected:

```
>RUN

NOW EXECUTING ROM #1
NOW EXECUTING ROM #3
EXECUTING ROM #1 AGAIN

READY
>
```

CALL 73: Battery-Backed RAM Disable

Use CALL 73 to disable the battery-backed RAM and purge reset. You see “Battery Backup Disabled” when you execute this call. The next power loss destroys the contents of RAM. When you reapply power (if JW7 is enable, page 1 -9), RAM is cleared and battery back-up is automatically re-enabled.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 73
```

Example

```
>CALL 73
```

CALL 74: Battery-Backed RAM Enable

CALL 74 enables the battery-backed RAM. You see “Battery Backup Enabled” when you execute this call. The battery is enabled on power-up and remains enabled until you execute a CALL 73 or until the battery fails.

Make sure jumper JW7 is set to the “battery enabled” position (page 1 -9).

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 74
```

Example

```
>CALL 74
```

CALL 75 – 76

Undefined. If you execute an undefined call, you receive the error message, “ERROR-UNSUPPORTED CALL”.

CALL 77: Protected Variable Storage

Use CALL 77 to reserve the top of RAM memory for protected variable storage. Values are saved if you enabled CALL 74 (page13 -5). You store values with the ST@ (page11 -35) statement and retrieve them with the LD@ (page11 -18) statement. Each variable you store requires 6 bytes of storage space.

You must subtract 6 times the number of protected variables you are storing from MTOP. This reduces the available RAM memory. PUSH this value onto the stack as the new MTOP address. All appropriate variable pointers are reconfigured. Do this only in Command mode to ensure proper operation.

Important:

- Use CALL 77 from Command mode only.
 - Do not let the ST@ address write over the MTOP address. This could alter the value of a variable or string.
 - CALL 77 de-allocates all the string memory along with all the string contents. Therefore, make sure you perform this call before execution of the string statement.
-

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the new MTOP address.

Syntax

```
PUSH new MTOP address
CALL 77
```

Example

```
>PRINT MTOP
14335
>PRINT MTOP-12 (2 VARIABLES TIMES 6 BYTES EACH)
14323
>PUSH 14323 (NEW MTOP ADDRESS)
>CALL 77
>10 K = 678 * PI
>20 L = 520
>25 PUSH K
>30 ST@ 14335:REM STORE K IN PROTECTED AREA
>40 PUSH L
>50 ST@ 14329
>55 REM TO RETRIEVE PROTECTED VARIABLES
```



```
>60 LD@ 14335:REM REMOVE K FROM PROTECTED AREA
>70 POP K
>80 LD@ 14329
>90 POP L
>100 REM USE LD@ AFTER POWER LOSS AND BATTERY BACK-UP IS USED
```

Using Protected Variable Storage Area

```
>PRINT MTOP
14335
>PRINT MTOP-24
14311
>PUSH 14311 (NEW MTOP ADDRESS)
>CALL 77
>90 M1=14335 : REM BEGIN STORING HERE
>100 PUSH A, B, C, D
```

Using the ST@ and LD@ Commands in a DO Loop

```
>200 DO
>210 ST@ M1
>220 M1=M1-6 :REM EACH VARIABLE = 6 BYTES
>230 UNTIL M1=MTOP : REM YOU DEFINED THE NEW MTOP W/CALL 77
>290 M1 = 14335
>300 DO
>310 LD@ M1
>320 M1 = M1-6
>330 UNTIL M1 = MTOP
>360 POP A, B, C, D
>370 PRINT A, B, C, D
```

Using an Array To Set Up the Data

```
>PRINT MTOP
14335
>PRINT MTOP-24
14311
>PUSH 14311 (NEW MTOP ADDRESS)
>CALL 77
>100 DIM A(4)
>110 DATA 10, 20, 30, 40
>120 FOR I = 1 TO 4 : READ A(I) : NEXT I
```

CALL 78: Set Program Port Communication Rate

Use CALL 78 to change the program port communication rate from its default value (1200 bit/s) to one of the following: 300, 600, 1200, 2400, 4800, 9600 or 19200 bit/s. The default communication rate for the program port is 1200 bit/s if port PRT1 is configured as the program port or 19200 bit/s if port DH485 is configured as the program port. PUSH the desired communication rate and CALL 78. The program port remains at this communication rate unless CALL 73 (page 13 -5) is invoked or one of these conditions is met:

- battery is dead or has been removed
- battery-backup capacitor is discharged
- if a PROG1 or PROG2 is executed and the EEPROM is removed or not programmed
- power is cycled

If this happens the communication rate of ports PRT1 and PRT2 defaults to 1200 bit/s and the DH485 port defaults to 19200 bit/s.

See also the MODE statement (page 11 -20)

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is the communication rate you want (300, 600, 1200, 2400, 4800, 9600 or 19200 bit/s).

Syntax

```
PUSH desired communication rate  
CALL 78
```

Example

```
>10 PUSH 4800  
>20 CALL 78
```

CALL 79: Set the Active LED Blinking State (No Operation)

This call does nothing. The active LED is supposed to blink while the BASIC module is in Command mode and remain solid while the BASIC module is in Run mode.

CALL 80: Check Battery Condition



Use CALL 80 to check the module's battery condition.

Refer to Chapter 3 for information on how to change the battery.

Important: Refer to Guidelines for Handling Lithium Batteries (publication number AG-5.4 to properly dispose of the lithium battery.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument is the status of the battery:

- 0 = battery is okay
- 1 = battery is low

Syntax

```
CALL 80  
POP battery status
```

Example

```
>10 CALL 80  
>20 POP C  
>30 IF C <>0 THEN PRINT "BATTERY LOW"  
>40 END
```

CALL 81: User PROM Check and Description

Use CALL 81 before storing a program in the EEPROM memory. This routine:

- determines the number of memory module programs
- determines the number of bytes left in the memory module
- determines the number of bytes in the RAM program
- prints a message indicating if enough space is available in the memory module for the RAM program
- checks memory module checksum if program is found
- prints a caution message if checksum fails

Important: CALL 81 cannot detect a defective memory module.

Input and Output Arguments

This routine has no input arguments and no output arguments.

Syntax

```
CALL 81
```

Example

```
>CALL 81
Number of BASIC programs in (E)EPROM..... 3
Available bytes to end of user (E)EPROM..... 7944
Available bytes to beginning of assembly pgm.. 3848
Length of BASIC program in RAM..... 76

Program will fit in (E)EPROM.

READY
>
```

CALL 82: Check User Memory Module Map

Use CALL 82 to check the user PROM and display a map of where all the BASIC programs are stored. Use this routine as an aid for assembly language programming. With this call you can determine where the empty space in the memory module is located and how much space is available.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 82
```

Example

```
>CALL 82  
  
8010H -- 805CH --> ROM 1  
805DH -- 80A9H --> ROM 2  
80AAH -- 80F6H --> ROM 3  
80F7H -- FFFFH --> UNUSED
```

CALL 83: Display DH485 Port Parameters

Use this routine to display the current DH485 port configuration on the terminal. Enter CALL 83 from the Command mode.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 83 
```

Example

```
>CALL 83  
19200 BAUD  
Host Node Address = 0  
Module Node Address = 1  
Maximum Node Address = 31
```

CALL 84: Transfer DH-485 Common Interface File to BASIC Input Buffer

Use CALL 84 to transfer up to 40 words starting at the designated offset of the DH-485 Common Interface File to the BASIC module input buffer starting at the same designated offset from word 0.

This call does not interrupt nor is it interrupted by a DH-485 read or write from or to the DH-485 common interface file.

Input and Output Arguments

This routine has two input arguments and one output argument. The first input argument is the starting offset in the DH-485 Common Interface File and the BASIC module input buffer (100 to 139). The second input argument is the length in words to be transferred (1 to 40). The output argument is the status of the call:

- 0 = successful transfer
- 1 = illegal starting offset; transfer does not take place
- 2 = illegal length; transfer does not take place

Syntax

```
PUSH starting word offset in DH-485 interface file  
PUSH number of words to be transferred  
CALL 84  
POP transfer status
```

Example

```
>1  REM EXAMPLE PROGRAM  
>40 PUSH 0 : REM OFFSET ADDRESS = 0  
>50 PUSH 32 : REM WORD OFFSET = 32  
>60 CALL 84 : REM TRANSFER THE DATA TO THE BASIC INPUT BUFFER  
>70 POP R : REM GET THE OUTPUT ARGUMENT  
>80 IF (R<>0) THEN PRINT "TRANSFER ERROR CODE = ",R  
>90 REM PRINT ERROR
```

CALL 85: Transfer BASIC Output Buffer to DH-485 Common Interface File

Use CALL 85 to transfer up to 40 words starting at the designated offset of the BASIC output buffer to the DH-485 Common Interface File starting at the same designated offset from word 0.

This call does not interrupt nor is it interrupted by a DH-485 read or write from or to the DH-485 common interface file.

Word integrity is guaranteed during this transfer. File integrity is not.

Input and Output Arguments

This routine has two input arguments and one output argument. The first input argument is the starting word offset (0 – 127) of the DH-485 Common Interface File and the BASIC output buffer. The second input argument is the length in words (1 to 40) to be transferred from the BASIC module output buffer to the DH-485 Common Interface File.

The output argument specifies the transfer status. It can have one of these values:

- 0 = successful transfer
- 1 = illegal starting offset; transfer does not take place
- 2 = illegal length; transfer does not take place

Syntax

```
PUSH starting word offset in DH-485 interface file
PUSH number of words to be transferred
CALL 85
POP transfer status
```

Example

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 31 : REM OFFSET ADDRESS = 31
>50 PUSH 3 : REM WORD LENGTH = 3
>60 CALL 85 : REM TRANSFER DATA TO DH-485 CIF
>70 POP R
>80 IF R<>0 PRINT "TRANSFER ERROR CODE = ",R
>90 REM PRINT ERROR

READY
>RUN

READY
```

CALL 86: Check DH-485 Interface File Remote Write Status

Use CALL 86 to determine if the DH-485 Common Interface File located in the BASIC module was updated since the last time you checked.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument is the DH-485 interface file remote write status:

- 0 – a device on the DH-485 Serial Communications Link has not written to the DH-485 Serial Common Interface File since the last time you executed this call or since you powered up the BASIC module, whichever occurred last.
- 1 – a device on the DH-485 Serial Communications Link has written to the DH-485 Common Interface File since the last time you executed this call or since you powered up the BASIC module, whichever occurred last.

Syntax

```
CALL 86  
POP DH-485 interface file remote write status
```

Example

```
>100 CALL 86 : REM CHECK FILE STATUS  
>110 POP X : REM GET THE STATUS  
>120 IF(X<>1) THEN GOTO 100 : REM WAIT ON THE DATA
```


CALL 87: Check DH-485 Interface File Remote Read Status

Use CALL 87 to determine if the DH-485 Common Interface File located in the BASIC module was read by a device on the DH-485 Serial Communications Link since the last time you checked.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument is the DH-485 interface file remote read status:

- 0 – a device has not read from the DH-485 Common Interface File since the last time you executed this call or since you powered up the BASIC module, whichever occurred last.
- 1 – a device on the DH-485 Serial Communications Link has read the DH-485 Common Interface File since the last time you executed this call or since you powered up the BASIC module, whichever occurred last.

Syntax

```
CALL 87  
POP DH-485 interface file remote read status
```

Example

```
>1   REM EXAMPLE PROGRAM  
>100 CALL 87 : REM CHECK FILE STATUS  
>110 POP X : REM GET THE STATUS  
>120 IF (X<>1) GOTO 100: REM WAIT ON DATA TO BE READ
```

CALL 88: BASIC Floating Point to PLC-5 Floating Point



Use this call to convert BASIC floating point to PLC-5 floating point in a two-word format and place the converted value in the block transfer read buffer. See also CALL 89.

See Chapter 8 for more information.

The BASIC module floating point number is an 8-digit BCD floating point number. The range of the BASIC module floating point number is;

$$\pm 1E^{-127} \text{ to } \pm 99999999E^{+127}$$

The PLC-5 floating point number is a 7-digit binary floating point number (IEEE Float 32-bit value). The range of the PLC-5 floating point number:

$$\pm 1.1754944E^{-38} \text{ to } \pm 3.4028237E^{+38}$$

The BASIC module has a floating point range larger than the floating point range of the PLC-5 processor. If CALL 88 attempts to convert a number larger than $\pm 3.4028237E^{+38}$, the converted number is assigned a value of $\pm 3.4028237E^{+38}$. If CALL 88 attempts to convert a number smaller than $\pm 1.1754944E^{-38}$, the converted number is assigned a value of $\pm 1.1754944E^{-38}$.

PLC-5 floating point numbers are stored in 2 words of the BTR buffer

Important: Due to the fact that the PLC-5 floating point number is a 7-digit floating point number, and the BASIC module is an 8-digit floating point number, some round off error may be introduced during number conversions.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input argument is the floating point value you want to convert. The second input argument is the first word in the BTR buffer (1 to 63) to receive the converted value.

Syntax

```
PUSH number to convert
PUSH output buffer to receive converted value
CALL 88
```

Example

```
>20 PUSH V :REM V F.P. NUMBER TO CONVERT
>30 PUSH 1 :REM WORD 1 and 2 of BTR BUFFER GETS A VALUE OF V
>40 CALL 88 :REM CONVERT VALUE
```

CALL 89: PLC-5 Floating Point to BASIC Floating Point



Use this call to convert PLC-5 floating point to BASIC floating point. See also CALL 88.

See Chapter 8 for more information.

The PLC-5 floating point number is a 7-digit binary floating point number (IEEE Float 32-bit value). The range of the PLC-5 floating point number:

$$\pm 1.1754944E^{-38} \text{ to } \pm 3.4028237E^{+38}$$

The BASIC module floating point number is an 8-digit BCD floating point number. The range of the BASIC module floating point number is:

$$\pm 1E^{-127} \text{ to } \pm 99999999E^{+127}$$

PLC-5 floating point numbers are stored in 2 words of the BTW buffer

Important: Due to the fact that the PLC-5 floating point number is a 7-digit floating point number, and the BASIC module is an 8-digit floating point number, some round off error may be introduced during number conversions.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the number (1 to 63) of the first word (PLC-5 floating point is sent to the BASIC module in two processor words) in the BTW buffer you want to convert. The output argument is the converted value.

Syntax

```
PUSH number of word to convert
CALL 89
POP converted value
```

Example

```
>50 PUSH 10 :REM CONVERT 10th and 11th WORD OF BTW BUFFER
>60 CALL 89 :REM CONVERT VALUE
>70 POP W :REM GET CONVERTED VALUE-STORE IN VARIABLE W
```

CALL 90: Read Remote DH-485 Data File to BASIC Input Buffer

Use CALL 90 to read up to 40 words from the designated node address, file number, file type, and element offset of a remote DH-485 data file to the BASIC module input buffer starting at word 100.

Input and Output Arguments

This routine has six input arguments and one output argument.

Argument	Description	Page
input 1	node address of the remote device (0–31)	13 -18
input 2	file number of the remote device (0–255) to be read	13 -18
input 3	file type read from the remote device	13 -18
input 4	starting element offset within the file on the remote device (0 to 32767)	13 -19
input 5	number of elements to be transferred	13 -19
input 6	message time-out value	13 -19
output 1	call status	13 -20

Input Argument One

The first input argument is the node address of the remote device (0 to 31). If the number is not within the range 0 to 31, then the output argument equals 10, and the read message does not take place.

Input Argument Two

The second input argument is the file number on the remote device (0 to 255) to be read. If the number is not within the range 0 to 255, then the output argument equals 11, and the read message does not take place.

Input Argument Three

The third input argument is the file type read from the remote device. If the file type is not one of the valid types listed in the table, then the output argument equals 241, and the read message does not take place.

File type	File type code	Words/Element
integer file	ASC(N)	1 word/element
status file	ASC(S)	1 word/element
counter file	ASC(C)	3 words/element
timer file	ASC(T)	3 words/element
bit file	ASC(B)	1 word/element
control file	ASC(R)	3 words/element

Input Argument Four

The fourth input argument is the starting element offset within the file on the remote device (0 to 32767). If the number is not within the range 0 to 32767, then the output argument equals 12, and the transfer does not take place.

Important: The offset is twice of what is expected. For example, if an offset of 3 is PUSHed, the data is written to the remote DH-485 data file beginning at element 6.

Input Argument Five

The fifth input argument is the number of elements to be transferred. If the number is not within the valid length range specified in the table, then the output argument equals 13, and the transfer does not take place.

File type code	Valid length range
ASC(N)	1 to 40
ASC(S)	1 to 40
ASC(C)	1 to 13
ASC(T)	1 to 13
ASC(B)	1 to 40
ASC(R)	1 to 13

Input Argument Six

The sixth input argument is the message time-out value. This value is the number of hundreds of milliseconds that are allowed to receive the read response (1 to 50 = 0.1 to 5.0 seconds). If the read response is not received within this time, the message aborts with the output argument equal to 55. If the number is not within the range 1 to 50, the output argument equals 14, and the transfer does not take place.

The read data from the remote device is read into the BASIC module input buffers starting at word 100 and filling as many words as specified by the element length of the message.

Output Argument One

The output argument specifies the status of the message instruction. Upon return from the call, the output argument has this definition:

Decimal output	Hexadecimal output	Description
0	00	successful completion
2	02	target node cannot accept the message at this time
3	03	target node cannot respond because message is too large
4	04	target node cannot respond because it does not understand the command parameters
5	05	BASIC module is off-line (not on link)
6	06	target node cannot respond because requested function is not available
7	07	target node does not respond
10	0A	BASIC module detects illegal target node address
11	0B	BASIC module detects illegal file number
12	0C	BASIC module detects illegal target file element offset
13	0D	BASIC module detects illegal target file length
14	0E	BASIC module detects illegal time-out value
16	10	target node cannot respond because of incorrect command parameters or unsupported command
55	37	message timed out (time-out value exceeded)
80	50	target node is out of memory
96	60	target node cannot respond because file is protected
231	E7	target node cannot respond because length requested is too large
235	EB	target node cannot respond because target node denies access
236	EC	target node cannot respond because requested function is currently unavailable
241	F1	BASIC module detects illegal target file type
250	FA	target node cannot respond because another node is file owner (has sole file access)
251	FB	target node cannot respond because another node is program owner (has sole access to all files)

Syntax

PUSH remote device node address
PUSH remote device file number
PUSH remote device file type
PUSH starting element offset (x2) of remote device file
PUSH number of elements to be transferred
PUSH message time-out value
CALL 90
POP status of message instruction

Example

```
>10 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>20 PUSH 5 : REM REMOTE FILE    5
>30 PUSH ASC(C) : REM FILE TYPE = COUNTER
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM ELEMENT LENGTH = 10 = 30 WORDS
>60 PUSH 5 : REM TIME-OUT = 0.5 SECONDS
>70 CALL 90
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF (R<>0) THEN PRINT "READ ERROR CODE =",R
```

READY

>RUN

READ ERROR CODE = 5

CALL 91: Write BASIC Output Buffer to Remote DH-485 Data File

Use CALL 91 to write up to 40 words starting at word 100 of the BASIC module output buffer to the remote DH-485 data file at the designated node address, file number, file type, and element offset.

Input and Output Arguments

This routine has six input arguments and one output argument.

Argument	Description	Page
input 1	node address of the remote device (0–31)	13 -22
input 2	file number on the remote device (0–255)	13 -22
input 3	file type written to the remote device	13 -22
input 4	starting element offset within the file on the remote device (0 to 32767)	13 -23
input 5	number of elements to be transferred	13 -23
input 6	message time-out value	13 -23
output 1	call status	13 -24

Input Argument One

The first input argument is the node address of the remote device (1 to 31). If the number is not within the range 1 to 31, then the output argument equals 10, and the write message does not take place.

Input Argument Two

The second input argument is the file number on the remote device (0 to 255). If the number is not within the range 0 to 255, then the output argument equals 11, and the write message does not take place.

Input Argument Three

The third input argument is the file type written to the remote device. If the file type is not one of the valid types listed in the table, then the output argument equals 241, and the write message does not take place.

File type	File type code	Words/Element
integer file	ASC(N)	1 word/element
status file	ASC(S)	1 word/element
counter file	ASC(C)	3 words/element
timer file	ASC(T)	3 words/element
bit file	ASC(B)	1 word/element
control file	ASC(R)	3 words/element

Input Argument Four

The fourth input argument is the starting element offset within the file on the remote device (0 to 255). If the number is not within the range (0 to 255), then the output argument equals 12, and transfer does not take place.

Important: The offset is twice of what is expected. For example, if an offset of 3 is PUSHed, the data is written to the remote DH-485 data file beginning at element 6.

Input Argument Five

The fifth input argument is the number of elements to be transferred. If the number is not within the range specified below, then the output argument equals 13, and the transfer does not take place.

File type	Valid length range
ASC(N)	1 to 40
ASC(S)	1 to 40
ASC(C)	1 to 13
ASC(T)	1 to 13
ASC(B)	1 to 40
ASC(R)	1 to 13

Input Argument Six

The sixth input argument is the message time-out value. This value is the number of hundreds of milliseconds that are allowed to receive the write response (1 to 50 = 0.1 to 5.0 seconds). If the write response is not received within this time, the message aborts with the output argument equal to 55. If the number is not within the range 1 to 50, the output argument equals 14, and the transfer does not take place.

The write data from the BASIC module output buffer is written to the remote device starting at word 100 and filling as many words as specified by the element length of the message.

Output Argument One

The output argument specifies the status of the message instruction. Upon return from the call, the output argument has this definition:

Decimal output	Hexadecimal output	Description
0	00	successful completion
2	02	target node cannot accept the message at this time
3	03	target node cannot respond because message is too large
4	04	target node cannot respond because it does not understand the command parameters
5	05	BASIC module is off-line (not on link)
6	06	target node cannot respond because requested function is not available
7	07	target node does not respond
10	0A	BASIC module detects illegal target node address
11	0B	BASIC module detects illegal file number
12	0C	BASIC module detects illegal target file element offset
13	0D	BASIC module detects illegal target file length
14	0E	BASIC module detects illegal time-out value
16	10	target node cannot respond because of incorrect command parameters or unsupported command
55	37	message timed out (time-out value exceeded)
80	50	target node is out of memory
96	60	target node cannot respond because file is protected
231	E7	target node cannot respond because length requested is too large
235	EB	target node cannot respond because target node denies access
236	EC	target node cannot respond because requested function is currently unavailable
241	F1	BASIC module detects illegal target file type
250	FA	target node cannot respond because another node is file owner (has sole file access)
251	FB	target node cannot respond because another node is program owner (has sole access to all files)

This call is implemented as a Protected Typed Logical Write with two address fields.

Refer to the DF1 Protocol and Command Set Reference Manual (publication number 1770-6.5.16) for detailed information on DH-485.



Syntax

PUSH remote device node address
PUSH remote device file number
PUSH remote device file type
PUSH starting element offset (x2) of remote device file
PUSH number of elements to be transferred
PUSH message time-out value
CALL 91
POP status of message instruction

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>20 PUSH 7 : REM REMOTE FILE 7
>30 PUSH ASC(N) : REM FILE TYPE = INTEGER
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10: REM WORD LENGTH = 10
>60 PUSH 5 : REM THE TIME-OUT VALUE = 0.5 SECOND
>70 CALL 91 : REM WRITE DATA FROM OUTPUT BUFFER
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF R<>0 PRINT "READ ERROR CODE =",R
```

READY

>RUN

READ ERROR CODE = 5

CALL 92: Read Remote DH-485 Common Interface File to BASIC Input Buffer

Use CALL 92 to read up to 40 words from the remote DH-485 Common Interface File of the designated node address, starting at the designated word offset to the BASIC module input buffer starting at word 100.

Input and Output Arguments

This routine has four input arguments and one output argument.

Argument	Description	Page
input 1	node address of the remote device (1–31)	13 -26
input 2	starting word offset within the file on the remote device (0 to 255)	13 -26
input 3	number of words to be transferred	13 -26
input 4	message time-out value	13 -26
output 1	call status	13 -27

Input Argument One

The first input argument is the node address of the remote device (1 to 31). If the number is not within the range 1 to 31, then the output argument equals 10, and the read message does not take place.

Input Argument Two

The second input argument is the starting word offset within the file on the remote device (0 to 255). If the number is not within the range 0 to 255, then the output argument equals 12, and the transfer does not take place.

Important: The offset is twice what you expect. For example, if an offset of 3 is PUSHed, the data is written to the remote DH-485 data file beginning at element 6.

Input Argument Three

The third input argument is the number of words to be transferred. If the number is not within the range (1 to 40), then the output argument equals 13, and the transfer does not take place.

Input Argument Four

The fourth input argument is the message time-out value. This value is the number of hundreds of milliseconds that are allowed to receive the read response (1 to 50 = 0.1 to 5.0 seconds). If the read response is not received within this time, the message aborts with the output argument equal to 55. If the number is not within the range 1 to 50, the output argument equals 14, and the transfer does not take place.

The read data from the remote device is read into the BASIC module input buffer starting at word 100 and filling as many words as specified by the word length of the message.

Output Argument One

The output argument specifies the status of the message instruction. Upon return from the call, the output argument has these definitions:

Decimal output	Hexadecimal output	Description
0	00	successful completion
2	02	target node cannot accept the message at this time
3	03	target node cannot respond because message is too large
4	04	target node cannot respond because it does not understand the command parameters
5	05	BASIC module is off-line (not on link)
6	06	target node cannot respond because requested function is not available
7	07	target node does not respond
10	0A	BASIC module detects illegal target node address
11	0B	BASIC module detects illegal file number
12	0C	BASIC module detects illegal target file element offset
13	0D	BASIC module detects illegal target file length
14	0E	BASIC module detects illegal time-out value
16	10	target node cannot respond because of incorrect command parameters or unsupported command
55	37	message timed out (time-out value exceeded)
80	50	target node is out of memory
96	60	target node cannot respond because file is protected
231	E7	target node cannot respond because length requested is too large
235	EB	target node cannot respond because target node denies access
236	EC	target node cannot respond because requested function is currently unavailable
241	F1	BASIC module detects illegal target file type
250	FA	target node cannot respond because another node is file owner (has sole file access)
251	FB	target node cannot respond because another node is program owner (has sole access to all files)

Syntax

PUSH remote device node address
PUSH starting element offset (x2) of remote device file
PUSH number of words to be transferred
PUSH message time-out value
CALL 92
POP status of message instruction

Example

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM WORD LENGTH = 10
>60 PUSH 5   REM TIME-OUT VALUE = 0.5 SECONDS
>70 CALL 92
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF (R<>0) THEN PRINT "READ ERROR CODE IS",R
>100 REM PRINT ERROR
```

READY

>RUN

READ ERROR CODE IS 5

CALL 93: Write Output Buffer to Remote DH-485 Common Interface File

Use CALL 93 to write up to 40 words starting at word 100 of the BASIC module output buffer to the remote DH-485 Common Interface File at the designated node address, starting at the designated word offset.

Input and Output Arguments

This routine has four input arguments and one output argument.

Argument	Description	Page
input 1	node address of the remote device (1–31)	13 -29
input 2	starting word offset within the file on the remote device (0 to 255)	13 -29
input 3	number of words to be transferred	13 -29
input 4	message time-out value	13 -29
output 1	call status	13 -30

Input Argument One

The first input argument is the node address of the remote device (1–31). If the number is not within the range 1 to 31, then the output argument equals 10, and the write message does not take place.

Input Argument Two

The second input argument is the starting word offset within the file on the remote device (0 to 255). If the number is not within the range (0 to 255), then the output argument equals 12, and the transfer does not take place.

Important: The offset is twice what you expect. For example, if an offset of 3 is PUSHed, the data is written to the remote DH-485 data file beginning at element 6.

Input Argument Three

The third input argument is the number of words to be transferred. If the number is not within the range specified below, then the output argument equals 13, and the transfer does not take place.

Input Argument Four

The fourth input argument is the message time-out value. This value is the number of hundreds of milliseconds that are allowed to receive the write response (1 to 50 = 0.1 to 5.0 seconds). If the write response is not received within this time, the message aborts with the output argument equal to 55. If the number is not within the range 1 to 50, the output argument equals 14, and the transfer does not take place.

The data from the BASIC module output buffer starting at word 100 is written to the remote common interface file starting at the specified word, and filling as many words as specified by the word length of the message.

Output Argument One

The output argument specifies the status of the message instruction. Upon return from the call, the output argument has this definition.

Decimal output	Hexadecimal output	Description
0	00	successful completion
2	02	target node cannot accept the message at this time
3	03	target node cannot respond because message is too large
4	04	target node cannot respond because it does not understand the command parameters
5	05	BASIC module is off-line (not on link)
6	06	target node cannot respond because requested function is not available
7	07	target node does not respond
10	0A	BASIC module detects illegal target node address
11	0B	BASIC module detects illegal file number
12	0C	BASIC module detects illegal target file element offset
13	0D	BASIC module detects illegal target file length
14	0E	BASIC module detects illegal time-out value
16	10	target node cannot respond because of incorrect command parameters or unsupported command
55	37	message timed out (time-out value exceeded)
80	50	target node is out of memory
96	60	target node cannot respond because file is protected
231	E7	target node cannot respond because length requested is too large
235	EB	target node cannot respond because target node denies access
236	EC	target node cannot respond because requested function is currently unavailable
241	F1	BASIC module detects illegal target file type
250	FA	target node cannot respond because another node is file owner (has sole file access)
251	FB	target node cannot respond because another node is program owner (has sole access to all files)

Syntax

PUSH remote device node address
PUSH starting element offset (x2) of remote device file
PUSH number of words to be transferred
PUSH message time-out value
CALL 93
POP status of message instruction

Example

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM WORD LENGTH = 10
>60 PUSH 5 : REM THE TIME-OUT VALUE = 0.5 SECONDS
>70 CALL 93 : REM WRITE DATA FROM BASIC OUTPUT BUFFER
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF R<>0 THEN PRINT READ ERROR CODE = ",R
```

READY

>RUN

READ ERROR CODE = 5

CALL 94: Display Current PRT1 Port Setup

Use CALL 94 to display the current PRT1 port configuration on the terminal screen.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 94 
```

Example

```
>CALL 94  
COMMUNICATION RATE= 9600  
DATA BITS= 8  
PARITY= NONE  
STOP BITS= 1  
HANDSHAKING= SOFTWARE
```

CALL 95: Get Number of Characters in PRT1 Buffers

Use CALL 95 to retrieve the number of characters in either the receive or transmit buffer of port PRT1.

Input and Output Arguments

This routine has one input and one output argument. The input argument is the buffer you want to examine:

- 0 for the transmit buffer
- 1 for the receive buffer

The output argument is the number of characters.

Syntax

```
PUSH buffer selection  
CALL 95  
POP number of characters
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 PUSH 0 : REM EXAMINES THE TRANSMIT BUFFER  
>20 CALL 95  
>30 POP X : REM GET THE NUMBER OF CHARACTERS  
>40 PRINT "NUMBER OF CHARACTERS IN PRT1 TRANSMIT BUFFER IS  
",X  
>50 END
```

CALL 96: Clear PRT1 Receive/Transmit Buffers

Use CALL 96 to clear port PRT1 receive and transmit buffers.

Input and Output Arguments

This routine has one input and no output argument. The input argument is the buffer you want to clear:

- 0 to clear the transmit buffer
- 1 to clear the receive buffer
- 2 to clear both buffers

Syntax

```
PUSH buffer selection  
CALL 96
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 PUSH 0 : CALL 96 : REM CLEAR PRT1 TRANSMIT BUFFER  
>30 END
```

CALL 97: Enable Port PRT2 DTR Signal

Use CALL 97 to enable the Data Terminal Ready (DTR) signal from port PRT2. The DTR signal is enabled by default when you power up the BASIC module. This call re-enables the DTR if it has been disabled by CALL 98.

Input and Output Arguments

This routine has no input and no output arguments.

Syntax

```
CALL 97
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 CALL 97 : REM ENABLE DTR SIGNAL  
>30 END
```

CALL 98: Disable Port PRT2 DTR Signal

Use CALL 98 to disable the Data Terminal Ready (DTR) signal from PRT2. CALL 97 re-enables the DTR signal.

Input and Output Arguments

This routine has no input and no output arguments.

Syntax

```
CALL 98
```

Example

```
>1  REM EXAMPLE PROGRAM  
>10 CALL 98 : REM DISABLE DTR SIGNAL  
>30 END
```

CALL 99: Reset Print Head Pointer

Use CALL 99 to reset the internal print head character counter of your printer when printing out wide forms. This call prevents the automatic CR/LF at character 79. You must keep track of the characters in each line.

Input and Output Arguments

This routine has no input and no output arguments.

Syntax

```
CALL 99
```

Example

```
>10 REM EXAMPLE PROGRAM  
>20 REM THIS PRINTS TIME BEYOND 80TH COLUMN  
>30 PRINT TAB(79)  
>40 CALL 99  
>50 PRINT TAB(41), "TIME -",  
>60 PRINT H,":",M,":",S  
>70 END
```

CALL 100: Download and Program Assembly Language Code to EEPROM

Use this call to store a file in user EEPROM through the program port of the BASIC module. The file must be in Intel Hex format. No checks are made on the addresses you are programming. An error message is generated to the program port if the EEPROM programming sequence fails. This call allows you to program assembly language programs in the BASIC module. The file (in Intel HEX Format) defines the address space in user EEPROM you can use for storage. This address space is controlled by “ORG” directives of the assembly language source. Further, Intel Hex Format defines an “END OF FILE” so the module knows when to stop.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 100
```

Example

```
>CALL 100
```

CALL 101: Upload User (E)EPROM Code to Host

Use this call to convert data within the address range to Intel Hex format, and print the information to the program port. An error message prints if the addresses are not consistent.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input is the starting address. The second input is the ending address.

Syntax

```
PUSH starting address  
PUSH ending address  
CALL 101
```

Example

```
>10 PUSH X  
>20 PUSH Y  
>30 CALL 101
```

CALL 102

Undefined. If you execute an undefined call, you receive the error message, “ERROR-UNSUPPORTED CALL.”

CALL 103: Print PRT1 Transmit Buffer and Pointer

Use CALL 103 to print the complete PRT1 transmit buffer with address, front pointer, and number of characters in the buffer to the console screen.

Use this information as a troubleshooting aid. It does not affect the contents of the buffer.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 103
```

Example

```
>CALL 103
```

PRT1 Output Queue

```
6D00H 3AH 31H 30H 38H 30H 34H 30H 30H 30H 39H 37H 34H 39H 30H 44H 30H
6D10H 41H 30H 30H 33H 43H 41H 30H 34H 41H 45H 41H 34H 46H 41H 36H 33H
6D20H 33H 30H 33H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H 38H
6D30H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 33H 48H 20H 34H 38H
6D40H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H 38H
6D50H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH 20H
6D60H 36H 44H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 34H 38H
6D70H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH 20H
6D80H 36H 44H 37H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 32H
6D90H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H
6DA0H 48H 20H 33H 34H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H
6DB0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 34H
6DC0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 32H
6DD0H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H
6DE0H 48H 20H 33H 34H 48H 0DH 0AH 20H 36H 44H 43H 30H 48H 20H 34H 38H
6DF0H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 38H 48H 20H 34H 34H
```

Output queue front pointer is: 6D29H

CALL 104: Print PRT1 Receive Buffer and Pointer

Use CALL 104 to print the complete PRT1 receive buffer with address, front pointer, and number of characters in the buffer to the console screen.

Use this information as a troubleshooting aid. It does not affect the contents of the buffer.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

CALL 104

Example

>CALL 104

PRT1 Input Queue

```
6C00H 33H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 7FH 7FH 7FH 7FH 7FH
6C10H 7FH 7FH 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 7FH 7FH
6C20H 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 0DH 0DH 0DH 0DH 0DH
6C30H 0DH 0DH 0DH 45H 58H 41H 4DH 7FH 7FH 7FH 7FH 52H 45H 4DH 20H 45H
6C40H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH 20H 50H 41H 47H 45H 20H
6C50H 36H 2DH 37H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 0DH 52H 4DH 41H
6C60H 4EH 54H 20H 7FH 7FH 7FH 54H 20H 44H 41H 54H 41H 0DH 52H 45H 4DH
6C70H 20H 54H 48H 45H 52H 45H 20H 49H 53H 20H 4EH 4FH 20H 52H 45H 41H
6C80H 4CH 20H 52H 45H 53H 50H 4FH 4EH 53H 45H 20H 57H 48H 49H 43H 48H
6C90H 20H 57H 49H 4CH 4CH 20H 53H 48H 4FH 57H 20H 55H 50H 20H 49H 4EH
6CA0H 20H 41H 4EH 20H 45H 58H 41H 4DH 50H 4CH 45H 0DH 0DH 0DH 0DH 0DH
6CB0H 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH 20H
6CC0H 50H 41H 47H 45H 20H 36H 2DH 36H 0DH 50H 55H 53H 48H 20H 38H 30H
6CD0H 30H 30H 48H 3AH 50H 7FH 7FH 20H 70H 55H 53H 7FH 7FH 7FH 3AH 20H
6CE0H 50H 55H 53H 48H 20H 38H 30H 7FH 30H 34H 46H 48H 20H 3AH 43H 41H
6CF0H 4CH 4CH 20H 31H 30H 31H 0DH 0DH 0DH 43H 41H 4CH 4CH 20H 31H 30H
```

Input queue front pointer is: 6C5DH

CALL 105: Reset PRT1 to Default Settings

Use this call to set the PRT1 port to 1200 bit/s, 8 bits, 1 stop bit, no parity, and software handshaking.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

CALL 105

Example

>CALL 105

CALL 106 – 107

Undefined. If you execute an undefined call, you receive the error message, “ERROR-UNSUPPORTED CALL.”

CALL 108: Enable DF1 Driver Communications

Use CALL 108 to enable DF1 driver communications via port PRT2. You can only enable this device if the operating mode jumper JW4 (page 1 -6) is in the correct position.

See CALL 113 (page 13 -47) to disable the DF1 driver.

Input and Output Arguments

This routine has six input arguments and no output arguments.

Argument	Description	Page
input 1	specifies the operational code selection that indicates the mode of operation for the DF1 driver	13 -38
input 2	specifies the Poll Time-out period when in half-duplex mode or the ACKnowledge Time-out period when in full-duplex mode	13 -42
input 3	number of message retries when in half-duplex mode or the number of ENQuery Retries to perform when in full-duplex mode	13 -43
input 4	RTS On Delay time period when in half-duplex mode or the number of NAK Received Retries to perform when in full-duplex mode	13 -43
input 5	RTS Off Delay time period	13 -43
input 6	BASIC module address that the DF1 driver responds to when receiving enquires from a remote DF1 device	13 -43

Input Argument One

The first input argument specifies the operational code selection that indicates the mode of operation for the DF1 driver.

The operational code specifies the following DF1 parameters:

- full-duplex or half-duplex slave operation
- duplicate packet detection (DPD) selection
- BCC or CRC error checking selection
- enable embedded responses (ER) or auto-detect embedded responses (ADER) (Ex. perform embedded responses only if embedded responses received form other station. Only applies to full-duplex operation).
- modem handshaking selection

Operational Codes for Half-Duplex Mode:

Operational code	Corresponding mode of operation	Special operational code (same as 0 - 11 except EOT is suppressed)
0	NHS, Disable DPD, BCC Error Checking	32
1	NHS, Enable DPD, BCC Error Checking	33
2	NHS, Disable DPD, CRC Error Checking	34
3	NHS, Enable DPD, CRC Error Checking	35
4	HDMwoCC, Disable DPD, BCC Error Checking	36
5	HDMwoCC, Enable DPD, BCC Error Checking	37
6	HDMwoCC, Disable DPD, CRC Error Checking	38
7	HDMwoCC, Enable DPD, CRC Error Checking	39
8	HDMwCC, Disable DPD, BCC Error Checking	40
9	HDMwCC, Enable DPD, BCC Error Checking	41
10	HDMwCC, Disable DPD, CRC Error Checking	42
11	HDMwCC, Enable DPD, CRC Error Checking	43

A special range of operational codes (32 - 43) are also accepted. These codes are identical to codes 0 - 11 except that the end of transmission (EOT) packets are suppressed. This operation is a deviation from the standard DF1 protocol and should only be used where transmissions from a slave module are minimized. When using one of these selections, the DF1 driver does not respond to ENQUIRES from a DF1 master unless there is a data packet transmitted.

Modem Handshaking for Half-Duplex Mode

Modem handshaking	Operational codes	Description
half-duplex no handshaking	0–3	<ul style="list-style-type: none"> • RTS output line is activated during transmission, but no RTS On Delay or RTS Off Delay is performed. • DTR output line is not manipulated by the DF1 driver. It is recommended that you activate DTR in your BASIC program while DF1 communications are taking place • CTS and DSR input lines are <i>not</i> monitored nor do they have any affect on transmissions or receptions. • Transmission monitor guarantees that transmitter interrupts are generated in a timely manner. If a time-out occurs as a data packet was transmitting, the DF1_Status is set to code value 5. Also, RTS is immediately dropped when this timeout occurs.
half-duplex without continuous carrier	4–7	<p>Important: For proper operation, connect the Data Carrier Detect (DCD) line from the modem to the DSR input of port PRT2.</p> <ul style="list-style-type: none"> • RTS output line is activated only during transmissions. The actual packet transmission starts after the delay specified by the RTS On Delay parameter, assuming the CTS input is active by then. When the transmission is complete and the delay time period specified by the RTS Off Delay parameter has timed out, RTS is deactivated. • Actual transmission does not start until the CTS input is active. A transmission guarantees that transmitter interrupts are generated in a timely manner. If a timeout occurs while a data packet is transmitting, then the DF1_Status is set to code value 5. RTS is dropped immediately when this occurs. • If not already active, the DTR line is raised when the DF1 Driver is enabled. Even after the DF1 Driver is disabled, it <i>will</i> remain active; you may deactivate it via CALL 98, page 13 -34. • Characters received only are accepted if the DCD line is active. A packet reception is aborted if DCD goes inactive during the byte-to-byte reception of that packet. There is no constant monitoring of DCD even between packets as there is with the constant carrier selection. Therefore, the DTR line is never deactivated.
half-duplex with continuous carries	8–11	<p>Important: For proper operation, connect the Data Carrier Detect (DCD) line from the modem to the DSR input of port PRT2.</p> <ul style="list-style-type: none"> • RTS output line is activated only during transmissions. The actual packet transmission starts after the delay specified by the RTS On Delay parameter, assuming the CTS input is active by then. When the transmission is complete and the delay time period specified by the RTS Off Delay parameter has timed out, RTS is deactivated. • Actual transmission does not start until the CTS input is active. A transmission guarantees that transmitter interrupts are generated in a timely manner. If a timeout occurs, then the DF1_Status is set to code value 5 if the data packet was being transmitted. RTS is dropped immediately when this occurs. • If not already active, the DTR line is raised when the DF1 Driver is enabled. It is dropped only when DCD is lost. Even after the DF1 Driver is disabled or remains active, you may deactivate it via CALL 98, page 13 -34. • For packet reception, the DCD signal is monitored (via the DSR input line). If DCD is not already active when the DF1 Driver is enabled, then it is immediately detected when it does go active. <p>At this point, the DCD is checked every 5 ms to make sure it remains active. If DCD goes inactive, the driver waits 10 seconds for it to go active again. If DCD does not go active again in this amount of time, then the DTR output line is dropped for a period of time ranging from 5 to 10 ms in length. Also, characters that are received are accepted if the DCD line is active. A packet reception is aborted if DCD goes inactive during the byte-to-byte reception of a packet.</p>

Operational Codes for Full-Duplex Mode:

Legal values for the operational code are 16 to 31 for full-duplex mode:

Operational code	Corresponding mode of operation
16	NHS, ER, Disable DPD, BCC Error Checking
17	NHS, ER, Enable DPD, BCC Error Checking
18	NHS, ER, Disable DPD, CRC Error Checking
19	NHS, ER, Enable DPD, CRC Error Checking
20	NHS, ADER, Disable DPD, BCC Error Checking
21	NHS, ADER, Enable DPD, BCC Error Checking
22	NHS, ADER, Disable DPD, CRC Error Checking
23	NHS, ADER, Enable DPD, CRC Error Checking
24	FDM, ER, Disable DPD, BCC Error Checking
25	FDM, ER, Enable DPD, BCC Error Checking
26	FDM, ER, Disable DPD, CRC Error Checking
27	FDM, ER, Enable DPD, CRC Error Checking
28	FDM, ADER, Disable DPD, BCC Error Checking
29	FDM, ADER, Enable DPD, BCC Error Checking
30	FDM, ADER, Disable DPD, CRC Error Checking
31	FDM, ADER, Enable DPD, CRC Error Checking

Important: Select other port parameters, such as communication rate, number of stop bits, and parity with the MODE command (page 11 -20) before you enable DF1. The modem handshaking selection made here overrides the handshaking parameter of the MODE command until DF1 is disabled.

Modem Handshaking for Full-Duplex Mode

Modem handshaking	Operational codes	Description
full-duplex with no handshaking	16–23	<ul style="list-style-type: none"> • RTS output line is activated when the DF1 Driver is enabled and remains so until the DF1 Driver is disabled. • DTR output line is not manipulated by the DF1 Driver. It is recommended that you activate DTR (CALL 97, page 13 -33) in your BASIC program while the DF1 communications is taking place. • CTS and DSR input lines are <i>not</i> monitored or have any effect on transmissions. • Transmission monitor guarantees that transmitter interrupts are generated in a timely manner. If a timeout occurs while a data packet is transmitting, then the DF1_Status is set to code value 5. RTS is <i>not</i> deactivated when this timeout occurs.
full-duplex modem (FDM)	24–31	<ul style="list-style-type: none"> • RTS output line is activated when the DF1 Driver is enabled and remains so until the DF1 Driver is disabled. • An actual transmission does not start until the CTS input is active. A transmission monitor guarantees that transmitter interrupts are generated in a timely manner. If a timeout occurs when a data packet is transmitting, then the DF1_Status is set to code value 5. RTS is <i>not</i> deactivated when this occurs. • If DTR is not already active when the DF1 Driver is enabled, it is immediately activated. It becomes active only if DCD is lost as described in the next paragraph. Even after the DF1 Driver is disabled, DTR remains active; you may deactivate it via CALL 98, page 13 -34. • For packet receptions, the DCD signal is monitored via the DSR input line. If DCD is not already active when the DF1 Driver is enabled, then it is immediately detected when it does go active. At this point, DCD is checked every 5 ms to make sure it remains active. If it goes inactive, the driver waits 10 seconds for DCD to go active again. If DCD does not go active again in this amount of time, then the DTR output line is deactivated for a period of time ranging from 5 to 10 ms in length. <p>Also, characters that are received are only accepted if the DCD line is active. A packet reception is aborted if DCD goes inactive during the byte-to-byte reception of that packet.</p>

Input Argument Two

The second input argument specifies the poll time-out period when in half-duplex mode or the ACKnowledge time-out period when in full-duplex mode. Poll time-out specifies in 5 ms increments how long to wait before being polled by the DF1 master, before a transmission request is ignored. PUSHing 0 indicates no poll time-out period. ACKnowledge time-out specifies in 5 ms increments how long to wait for an ACK/NAK before transmitting an ENquiry. The valid range for the ACKnowledge time-out is 2 to 65535.

Input Argument Three

The third input argument specifies the number of message retries when in half-duplex mode or the number of ENQuery retries to perform when in full-duplex mode. Message retries specifies the number of message transmission retry attempts made before giving up and flagging the transmission as failed. PUSHing 0 indicates only the initial attempt is made and if not acknowledged by the master the attempt is flagged as failed. ENQuery retries specifies the number of ENQ's to transmit before a packet transmission is flagged as failed. The valid range for both is 0 to 254.

Input Argument Four

The fourth input argument specifies the RTS On Delay time period when in half-duplex mode or the number of NAK received retries to perform when in full-duplex mode. RTS On Delay specifies in 5 ms increments the delay between when a Request-To-Send (RTS) is activated and a transmission is initiated. Only used if HDMwCC or HDMwoCC is selected through the first input argument. The valid range for the RTS On Delay is 0 to 65535. NAK Received Retries specifies the number of packet retries to transmit due to receiving NAK responses. The valid range for NAK Received Retries is 0 to 254.

Input Argument Five

The fifth input argument specifies the RTS Off Delay time period. RTS Off Delay specifies in 5 ms increments the delay between when a transmission is completed and a Request-To-Send (RTS) is deactivated. The valid range for the RTS Off Delay is 0 to 65499. This argument is only used if HDMwCC or HDMwoCC is selected through the first input argument. This input argument is only used for half-duplex mode. When full-duplex mode is selected a NULL value must be PUSHed.

Input Argument Six

The sixth input argument specifies the BASIC module address that the DF1 driver responds to when receiving enquires from a remote DF1 device. Legal values are 0 to 254. This input argument is used for half-duplex and full-duplex mode.

Syntax

PUSH operational code
PUSH poll timeout or ACKnowledge timeout
PUSH message retries or ENquiry retries
PUSH RTS On delay or NAK received retries
PUSH RTS Off delay or NULL value
PUSH BASIC module DF1 address
CALL 108

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 5 : REM HDMWOCC, ENABLE DPD, BCC ERROR CHECKING
>20 PUSH 200 : REM WAIT 1 SECOND TO BE POLLED BY MASTER
>30 PUSH 2 : REM PERFORM 2 RETRIES
>40 PUSH 4 : REM 20 MS RTS ON DELAY
>50 PUSH 4 : REM 20 MS RTS OFF DELAY
>60 PUSH 10 : REM BASIC MODULE ADDRESS OF 10
>70 CALL 108
>80 END
```

CALL 109: Print the Argument Stack

Use CALL 109 to prints the top 9 values on the argument stack to the console. Use this information as a troubleshooting aid. It does not affect the contents of or pointer to the stack.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

CALL 109

Example

```
>CALL 109

1C9H 00H 00H 00H 00H 00H 00H
1CFH 00H 00H 00H 00H 00H 00H
1D5H 00H 00H 00H 00H 00H 00H
1DBH 13H 04H 00H 00H 00H 7CH
1E1H 14H 42H 11H 96H 00H 83H
1E7H 11H 05H 92H 00H 00H 88H
1EDH 10H 00H 00H 00H 00H 84H
1F3H 20H 00H 00H 00H 00H 82H
1F9H 00H 00H 00H 00H 00H 00H
```

Argument stack pointer is: 01FEH

CALL 110: Print the PRT2 Port Transmit Buffer and Pointer

Use CALL 110 to print the complete PRT2 transmit buffer with addresses, front pointer and the number of characters in the transmit buffer to the console. Use this information as a troubleshooting aid—contents of the buffer are unaffected.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 110
```

Example

```
>CALL 110  
PRT2 Output Queue
```

```
6F00H 3AH 31H 30H 38H 30H 34H 30H 30H 30H 39H 37H 34H 39H 30H 44H 30H  
6F10H 41H 30H 30H 33H 43H 41H 30H 34H 41H 45H 41H 34H 46H 41H 36H 33H  
6F20H 33H 30H 33H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H 38H  
6F30H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 33H 48H 20H 34H 38H  
6F40H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H 38H  
6F50H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH 20H  
6F60H 36H 44H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 34H 38H  
6F70H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH 20H  
6F80H 36H 44H 37H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 32H  
6F90H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H  
6FA0H 48H 20H 33H 34H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H  
6FB0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 34H  
6FC0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 32H  
6FD0H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H 33H  
6FE0H 48H 20H 33H 34H 48H 0DH 0AH 20H 36H 44H 43H 30H 48H 20H 34H 38H  
6FF0H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 38H 48H 20H 34H 34H
```

```
Output queue front pointer is: 6F29H
```

CALL 111: Print the PRT2 Port Receive Buffer and Pointer

Use CALL 111 to print the complete PRT2 receive buffer with addresses, front pointer and the number of characters in the buffer to the console. Use this information as a troubleshooting aid—contents of the buffer are unaffected.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 111
```

Example

```
>CALL 111
```

```
PRT2 Input Queue
```

```
6E00H 33H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 7FH 7FH 7FH 7FH 7FH 7FH
6E10H 7FH 7FH 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 7FH 7FH
6E20H 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 0DH 0DH 0DH 0DH 0DH
6E30H 0DH 0DH 0DH 45H 58H 41H 4DH 7FH 7FH 7FH 7FH 52H 45H 4DH 20H 45H
6E40H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH 20H 50H 41H 47H 45H 20H
6E50H 36H 2DH 37H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 0DH 52H 4DH 41H
6E60H 4EH 54H 20H 7FH 7FH 7FH 54H 20H 44H 41H 54H 41H 0DH 52H 45H 4DH
6E70H 20H 54H 48H 45H 52H 45H 20H 49H 53H 20H 4EH 4FH 20H 52H 45H 41H
6E80H 4CH 20H 52H 45H 53H 50H 4FH 4EH 53H 45H 20H 57H 48H 49H 43H 48H
6E90H 20H 57H 49H 4CH 4CH 20H 53H 48H 4FH 57H 20H 55H 50H 20H 49H 4EH
6EA0H 20H 41H 4EH 20H 45H 58H 41H 4DH 50H 4CH 45H 0DH 0DH 0DH 0DH 0DH
6EB0H 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH 20H
6EC0H 50H 41H 47H 45H 20H 36H 2DH 36H 0DH 50H 55H 53H 48H 20H 38H 30H
6ED0H 30H 30H 48H 3AH 50H 7FH 7FH 20H 70H 55H 53H 7FH 7FH 7FH 3AH 20H
6EE0H 50H 55H 53H 48H 20H 38H 30H 7FH 30H 34H 46H 48H 20H 3AH 43H 41H
6EF0H 4CH 4CH 20H 31H 30H 31H 0DH 0DH 0DH 43H 41H 4CH 4CH 20H 31H 30H
```

```
Input queue front pointer is: 6E29H
```


CALL 112: User LED Control

Use CALL 112 to activate or de-activate the user LEDs (LED1 and LED2).

When you change to Command mode your user-defined LEDs remain in their last state until you re-enter Run mode.

Input and Output Arguments

This routine has two input arguments and no output arguments. The first input argument activates or de-activates LED1. The second input argument activates or de-activates LED2:

- 1 = activate LED
- 0 = deactivate LED

Any other value has no effect on that particular LED.

Syntax

```
PUSH LED1 state  
PUSH LED2 state  
CALL 112
```

Example

```
>100 PUSH 1 : REM TURN ON LED1  
>110 PUSH 0 : REM TURN OFF LED2  
>120 CALL 112 : REM SET THE LEDES
```

CALL 113: Disable DF1 Driver Communications

Use CALL 113 to disable DF1 driver communications. This call terminates DF1 communication immediately, even if the serial transmission of a data packet is in progress. You should write your user program so that it completes any transmission before performing CALL 113. This call clears the PRT2 transmit and receive buffers.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 113
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 CALL 113  
>20 END
```

CALL 114: Transmit DF1 Packet

Use CALL 114 to transmit the DF1 data packet. When you perform CALL 114, the DF1 data is posted for the DF1 driver to transmit as a single message packet. If you selected half-duplex slave operation, the message packet is transmitted the next time an ENquiry is received from the DF1 master. If you selected full-duplex operation, the message packet is transmitted immediately.

Use one or more PRINT# (page 11 -29), PH0.#, or PH1.# (page 11 -27) statements to construct the desired data in the transmit buffer of port PRT2. After constructing the data in the transmit buffer, use CALL 114 to initiate transmission of the data inside a DF1 message packet.

Use caution when building DF1 data packets. If you attempt to transmit five or less bytes of data (minimum is six bytes), the message ERROR: DF1 DATA PACKET TO TRANSMIT IS TOO SMALL is sent to the program port and the BASIC module enters Command mode.

If you attempt to place more than 256 bytes of data into the transmit buffer, the message ERROR: BUFFER OVERFLOW is sent to the program port and the BASIC module enters Command mode.

Your program must wait for one transmission to complete before performing construction of another data packet. Use CALL 115 to check the DF1 transmission status to determine when a transmission is complete.

Input and Output Arguments

This routine has no input or output arguments.

Syntax

```
CALL 114
```

Example

```
>1  REM EXAMPLE PROGRAM  
>10 CALL 114  
>20 END
```

CALL 115: Check DF1 Status

Use CALL 115 to check the DF1 transmit status.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument returns a value that represents the DF1 transmit status:

- 0 = no transmit result pending
- 1 = transmit result pending
- 2 = transmission successful
- 3 = transmission failed
- 4 = enquiry timeout, no transmission– this status should never be returned if full-duplex mode is selected
- 5 = if modem handshaking is selected, either a loss of CTS signal while transmitting or a fatal transmitter failure has occurred. If no handshaking is selected, a fatal transmitter failure has occurred
- 6 = if modem handshaking with constant carrier has been selected for either half-duplex or full-duplex modes, this error indicates transmission failure due to modem disconnection (DCD signal loss for more than 10 seconds)
- 7 = DF1 driver is not enabled

Syntax

```
CALL 115  
POP DF1 transmit status
```

Example

```
>1 REM EXAMPLE PROGRAM  
>10 CALL 115  
>20 POP X  
>30 END
```

CALL 116: Call User Defined Assembly Language Routine

Use this call to execute a user generated assembly language routine. This call performs some preliminary checks. If all the checks pass, then the user generated code is executed.

Input and Output Arguments

The number of input and output arguments are user defined. There must be at least one input argument. The last input argument must be the absolute address of the first byte of the header which precedes the routine. The user generated assembly language code must be preceded by a four byte header and followed by a two byte footer:

first header byte	must be a AAH
second header byte	must be a 55H
third header byte	absolute address of the first byte of footer
fourth header byte	absolute address of the first byte of footer
first footer byte	must be a 55H
second footer byte	must be a AAH

If any of the checks fail, then the stack is cleared, an error message is printed to the program port and the BASIC code execution stops.

Important: Programming the BASIC module with assembly language is extremely difficult and complex. Allen-Bradley does not support assembly language programming with the BASIC module. We recommend you instead use the C tool kit and C compiler available from one of our Pyramid Solutions Program Partners.

Syntax

```
PUSH address  
CALL 116
```

Example

```
>100 PUSH 8000H : CALL 116  
>110 POP a,B,C
```

CALL 117: Get DF1 Packet Length

Use CALL 117 to get the length of the DF1 data packet.

When CALL 117 is read in a program, the BASIC module checks to see if DF1 communications have been enabled through CALL 108 (page13 -38). If DF1 communications have not been enabled, an error message is printed to the console device and the BASIC module enters Command mode.

After you receive the length of the DF1 packet has been retrieved, you must use it in conjunction with the GET statement (page11 -12). The GET statement retrieves the data in the received DF1 packet.

Important: If the receive buffer is found empty, then 0000 is returned to the argument stack.

Input and Output Arguments

This routine has no input arguments and one output argument. The output argument returns the length of the oldest DF1 packet queued up in the DF1 receive buffer.

Syntax

```
CALL 117  
POP length of DF1 packet
```

Example

```
>1  REM EXAMPLE PROGRAM  
>10 CALL 117  
>20 POP X  
>30 END
```

CALL 118: PLC/SLC Unsolicited Writes

Important: This call requires the BASIC module jumper JW5 to be in 16 point mode (page 1 -7).

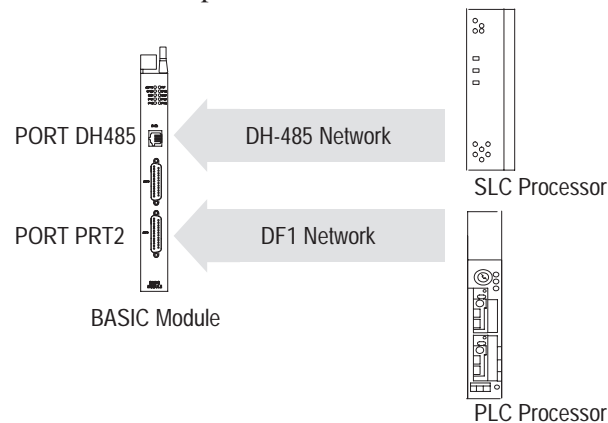
Use CALL 118 to allow the BASIC module to receive data packets sent by PLC-2, PLC-3, or PLC-5 message instructions on the DF1 network. This call also sets up the BASIC module to receive data packets from an SLC node on the DH-485 network. Both the DF1 port (PRT2) and the DH485 port cannot be active at the same time. Use jumper JW4 (page 1 -6) on the BASIC module to select your port configuration.

Any write message instruction sent to the BASIC module from these PLC/SLC processors causes the data to be placed in the BTR buffer and/or the BASIC module string, starting at the designated word offset.

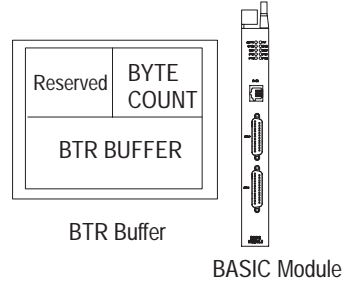
The low byte of word 1 of the BTR buffer contains the character count (byte count) of the transferred data. The high byte of this word is reserved. If you choose an internal string, the first character contains the byte count. The second character (transaction number) of the internal string is incremented, upon successful receipt of a packet, to inform the BASIC module that new data is in the string. The value of this transaction number wraps around from 255 to 0.

Execute CALL 118 once. After the call is executed, the BASIC module checks the port at the end of each line of BASIC code.

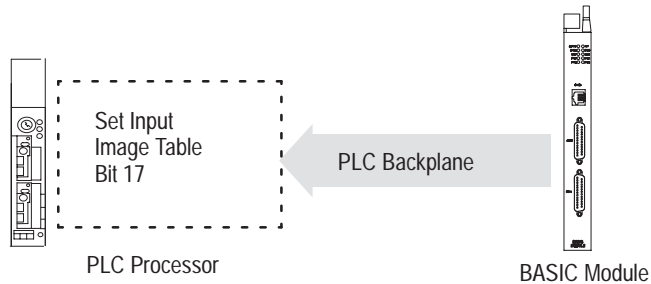
1. The BASIC module receives packets initiated from the PLC/SLC processor configured in the call through either ports PRT2 or DH485. PRT2 and DH485 ports cannot be active at the same time.



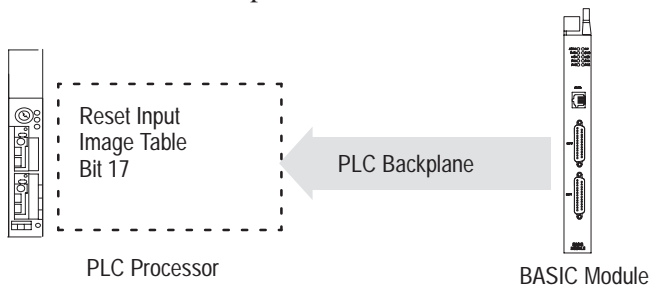
- The BASIC module transfers the data into the local PLC BTR buffer and places the byte count into the lower byte of the BTR word 1. The upper byte of BTR word 1 is reserved.



- The BASIC module sets the input image table bit 17 to inform the PLC processor that valid data is available. The BASIC module then initiates a block transfer read.



- The PLC processor retrieves the data from the block transfer. The BASIC module detects a successful block transfer and resets the input image table bit 17 on the same end of scan cycle in which the block transfer was performed.



This call is active until you re-execute it with different input parameters. If this occurs, the previous CALL 118 is automatically disabled and the new CALL 118 takes effect. You cannot execute multiple CALL 118s in parallel.

Input and Output Arguments

This routine has five input arguments and one output argument.

Argument	Description	Page
input 1	enables or disables the call	13 -54
input 2	BTR buffer, with or without the internal string, or the internal string alone	13 -54
input 3	always 1	13 -54
input 4	the string number	13 -55
input 5	maximum word length allowed for the data packet	13 -55
output 1	call status	13 -55

Input Argument One

The first input argument enables or disables the call:

- 0 = disable the previously executed CALL 118
- 1 = enable CALL 118. These commands are acceptable:
 - PLC (Unprotected writes)
 - PLC (Word range writes)
 - PLC (Typed writes)
 - SLC 5/02 (Unprotected writes)
 - SLC 5/02 (Typed writes)

If the data received exceeds the string length or CPU file size, the remaining data is truncated.

Input Argument Two

The second input argument is the selection of the BTR buffer with or without the internal string, or the internal string alone:

- 0 = BTR buffer
- 2 = internal string
- 4 = BTR buffer and internal string

If you choose the internal string (2), the input/output table data handshaking bit (17) is not used to indicate that data was received by the BASIC module. In the BASIC program, you must monitor the second character of the string (transaction number) which is incremented upon every successful data transfer. Then you must remove the data from the string before the next data packet is received or data is lost.

Input Argument Three

The third input argument is always 1.

Input Argument Four

The fourth input argument is the string number. If the second input argument does not select internal string usage, the value of this input argument is ignored but you must still PUSH it.

Input Argument Five

The fifth input argument is the maximum word length allowed for the data packet. Any packets received by the BASIC module of greater size are rejected. Entering 0 causes the BASIC module to accept packets of any size and all packets are received up to the maximum length of the destination file. Excess data is truncated.

Output Argument One

The output argument is the status of the call:

- 0 = successful
- 1 = disabled
- 2 = bad input parameter
- 3 = selected DH485/DF1 port not enabled
- 4 = string too small
- 5 = string is not dimensioned
- 6 = JW5 not in 16-point position

Syntax

```
PUSH CALL enable/disable  
PUSH selection of BTR file and/or string  
PUSH 1  
PUSH string number  
PUSH maximum word length  
CALL 118  
POP CALL 118 status
```

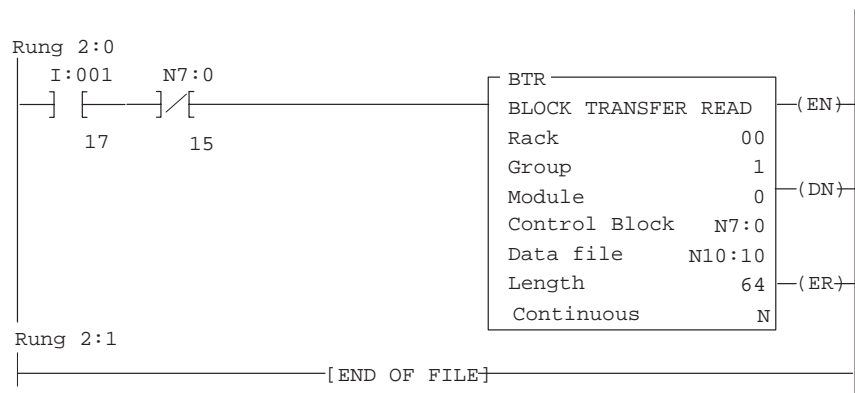
Example

```

>1  REM EXAMPLE PROGRAM
>10 REM ENABLE PLC/SLC UNSOLICITED WRITE INTERRUPT
>15 PUSH 64: CALL 4: REM SET BLOCK TRANSFER WRITE LENGTH
>16 PUSH 64: CALL 5: REM SET BLOCK TRANSFER READ LENGTH
>20 PUSH 1 : REM ENABLE THE CALL
>30 PUSH 0 : REM BTR BUFFER
>40 PUSH 1 : REM ALWAYS 1
>50 PUSH 0 : REM STRING NUMBER - NOT USED
>60 PUSH 20 : REM MAX ALLOWED WORD LENGTH OF DATA PACKET
>70 CALL 118
>80 POP S
>90 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 118 SETUP"

```

Sample Ladder Logic



CALL 119: Reset the PRT2 Port to Default Settings

Use CALL 119 to reset the PRT2 port to these default settings:

- 8 bits/character
- 1 stop bit
- no parity
- DCD off
- XON-XOFF off

Input and Output Arguments

This routine has no input and no output arguments.

Syntax

```
CALL 119
```

Example

```

>1  REM EXAMPLE PROGRAM
>10 CALL 119
>20 END

```

CALL 120: Clear BASIC Module I/O Buffers

Use this call to clear the BASIC module input and output buffers.

Input and Output Arguments

This routine has one input argument and no output arguments. The input argument is an 8-bit word that corresponds to the BASIC module input and output buffers shown here:

Bit	Decimal equivalent	BASIC module input and output buffer area
0		not used
1		not used
2	4	PLC BTW buffer
3	8	PLC BTR buffer
4	16	reserved
5	32	reserved
6		not used
7		not used

You must **PUSH** the decimal equivalent of the areas of the input and output buffers that you want to clear. For example to clear the BTW and BTR buffers you would push the value “12”. The BASIC module sets bits 2 and 3 true and clears the BTW and BTR buffers.

Important: CALL 120 does not clear the block-transfer buffers for background operations (CALLs 33, 34, 49, 50, 118, 122, and 123). The standard block-transfer buffers are cleared using the bits defined above.

Syntax

PUSH clear BTW and BTR buffers

CALL 120

Example

```
>1 REM EXAMPLE PROGRAM
>10 PUSH 12 :REM CLEAR BTW AND BTR BUFFERS
>20 CALL 120
>30 END
```

CALL 121

Undefined. If you execute an undefined call, you receive the error message, “ERROR-UNSUPPORTED CALL.”

CALL 122: Read Remote DF1 PLC Data File

Important: This call requires the BASIC module jumper JW5 to be in 16 point mode (page 1 -7).

Use CALL 122 to read up to 63 words of data from a remote DF1 node (PLC-2, -3, or -5) to the BTW buffer, and/or a string within the BASIC module.

This table lists specific notes when using CALL 122 with the PLC-3 and PLC-5 processor.

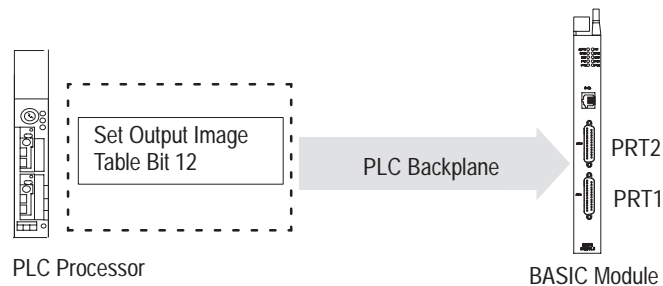
PLC processor	Notes
3	for timers and counters, the file number pushed (third parameter) is the structure number, limited to a maximum of 255 words
5	for timer data, an element is three 16-bit words, stored in the destination file in the following order: Control, Preset, and Accumulator

If you choose an internal string, the first character (transaction number) increments upon a successful read transaction to inform the BASIC module that new data is in the string. The value of the transaction number wraps around from 255 to 0.

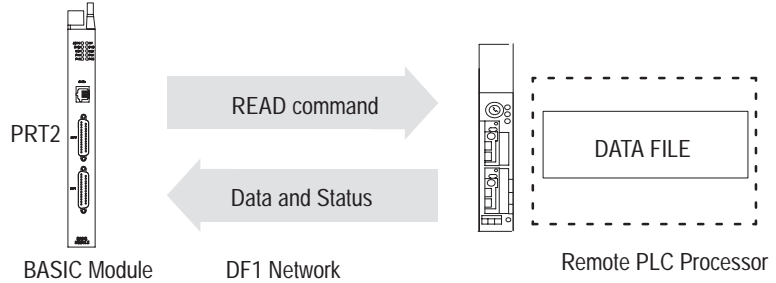
Set up the DF1 port parameters with CALL 108 (page13 -38). The DF1 port can operate with full-duplex or half-duplex slave protocol. Also make jumper JW4 is configured for DF1 protocol (page 1 -6).

Execute CALL 122 once to set up the data transfer parameters. PLC input table bit 12 and output table bit 12 is used to initiate and notify completion of transfer.

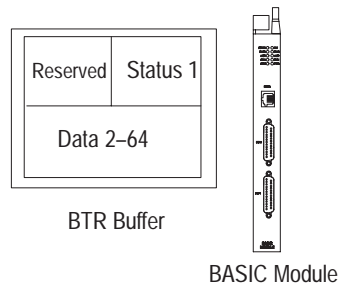
1. The local PLC processor sets the output image table bit 12 to inform the BASIC module to execute the READ command configured in this call.



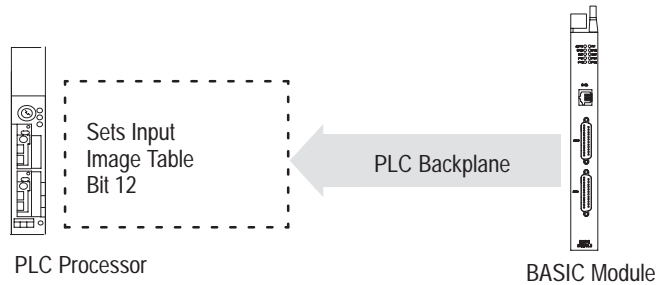
- The BASIC module issues the appropriate READ command to the remote PLC. The data and status are received from the PLC processor.



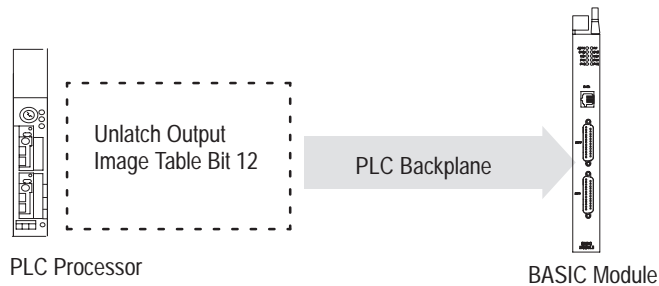
- When data is available, the BASIC module transfers the data into the BTR buffer and the BASIC module places the transaction status in the lower byte of BTR buffer word 1. The upper byte of BTR word 1 is reserved.



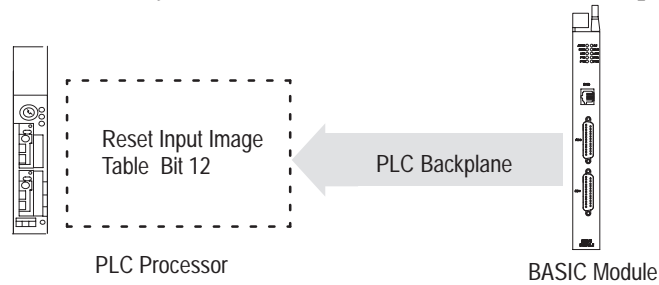
- The BASIC module sets the input image table, bit 12 and performs a block transfer read.



- The PLC receives the data and status from the block transfer and unlatches output image table bit 12 to inform the BASIC module that data was received.



- The BASIC module resets the input image table bit 12 on the same end of scan cycle in which the block transfer was complete.



This call is active until you re-execute it with different input parameters.

Input and Output Arguments

This routine has ten input arguments and one output argument.

Argument	Description	Page
input 1	type of PLC READ command issued	13 -61
input 2	node address of the remote PLC device (0-255)	13 -61
input 3	file number to be read on PLC remote device (0-255)	13 -61
input 4	file type to be read from the PLC remote device	13 -61
input 5	starting word offset within the file	13 -61
input 6	number of elements to be transferred	13 -62
input 7	message time-out value	13 -62
input 8	BTR buffer and/or internal string	13 -62
input 9	always 1	13 -62
input 10	string number	13 -62
output 1	call status	13 -63

To disable this call, PUSH a zero into the first input parameter. All other parameters are ignored but you must still PUSH them.

Input Argument One

The first input argument is the type of PLC READ command you issued:

- 0 = disable the previously executed CALL 122
- 2 = common interface file – PLC-2 unprotected READ command
- 3 = PLC-3 file – word range READ command
- 5 = PLC-5 file – typed READ command

Input Argument Two

The second input argument is the node address of the remote PLC device (0 through 255). If the number is not within this range, the status equals 2 and the read message does not occur.

Input Argument Three

The third input argument is the file number to be read on the PLC remote device (0 through 255). If the number is not within this range, the status equals 2 and the read message does not occur. The parameter is ignored if the common interface file is chosen in the first parameter, but you must still PUSH it.

Input Argument Four

The fourth input argument is the file type to be read from the PLC remote device. Enter the file type code as shown below. This argument is ignored if you chose the common interface file in the first parameter but you must still PUSH it (assumes integer type). If the file type is not one of these, the status equals 2 and the read message does not take place.

File type	File type code	Words/Element (1 word = 16 bits)
integer file	ASC(N)	1 word/element
status file	ASC(S)	1 word/element
counter file	ASC(C)	3 words/element
timer file	ASC(T)	3 words/element
bit file	ASC(B)	1 word/element
control file	ASC(R)	3 words/element
input file	ASC(I)	1 word/element
output file	ASC(O)	1 word/element

Input Argument Five

The fifth input argument is the starting word offset within the file on the PLC-2 remote device (0 through 32766). For PLC-3 integer, binary, or status files, the value is 0–9999. For PLC-3 I/O files, the value is 0–4095. For PLC-3 timer or counter files the value must be 0. If the number is not within this range, the status equals 2 and the transfer does not occur.

Input Argument Six

The sixth input argument is the number of elements to be transferred. If the number is not within the range listed in the table, the status equals 2 and the transfer does not occur.

File type code	Valid element length range
ASC(N)	1 to 63
ASC(S)	1 to 63
ASC(C)	1 to 21
ASC(T)	1 to 21
ASC(B)	1 to 63
ASC(R)	1 to 21
ASC(I)	1 to 21
ASC(O)	1 to 21
common interface file	1 to 21

Input Argument Seven

The seventh input argument is the message time-out value. This value (1 through 255) corresponds to the number of hundreds of milliseconds that are allowed to receive the read response (0.1 through 25.5 seconds). If the read response is not received within this time, the message aborts with the status equal to 55 in the input file word 1. If the time-out value is not within the range (1 through 255), the status output argument equals 2 and the transfer does not take place.

Input Argument Eight

The eighth input argument is the selection of the BTR buffer with or without the internal string or the internal string alone:

- 0 = BTR buffer
- 2 = internal string
- 4 = BTR buffer and internal string

If you chose internal string (2), execute CALL 29 (page 12-18) to initiate each data transfer without requiring PLC processor interaction. The output table bit 10 also initiates string transactions.

Input Argument Nine

The ninth input argument is always 1.

Input Argument Ten

The tenth input argument is the string number. If the eighth input argument does not select internal string usage, the value of this input argument is ignored, but you must still PUSH it.

Output Argument One

The output argument is the status of the call. It has these values:

- 0 = successful
- 1 = disabled
- 2 = bad input parameter
- 3 = DF1 not enabled
- 4 = string too small
- 5 = string is not dimensioned
- 6 = JW5 not in 16-point position

Whenever you attempt to read a remote node, the status of the read is placed into BTR word 1. The possible status codes are:

Code	Indicates
0	transfer OK
1	transmission failed
2	enquiry time out
3	<ul style="list-style-type: none"> • with handshaking selected – either a loss of CTS signal while transmitting or a fatal transmitter failure occurred • without handshaking selected – a fatal transmitter failure occurred
4	transmission failure due to modem disconnection (DCD signal loss for more than 10 seconds) if modem handshaking with constant carrier is selected
5	DF1 driver is not enabled
6	message timed out
81	illegal command or format
82	host has a problem and will not communicate
83	remote station host is not there, disconnected, or shut down
84	host could not complete function due to hardware fault
85	addressing problem or memory protect rungs
86	function disallowed due to command protection selection
87	processor is in Program mode
88	compatibility mode file missing or communication zone problem
89	remote station cannot buffer command
8B	remote station problem due to download
8C	local station cannot execute command due to active IPBs
C1	illegal address format – field has an illegal value
C2	illegal address format – not enough fields specified
C3	illegal address format – too many fields specified
C4	illegal address format – symbol not found
C5	illegal address format – symbol is 0 or greater than the maximum number of characters supported by this device
C6	illegal address – address does not exist or does not point to something usable in this command
C7	illegal size – file is wrong size; address is past end of file
C8	cannot complete request
C9	data or file is too large

Code	Indicates
CA	request is too large; transaction size plus word address is too large
CB	access denied, privilege violation
CC	resource is not available; condition cannot be generated
CD	resource is already available; condition already exists
CE	command cannot be executed
CF	overflow; histogram overflow
D0	no access
D1	illegal data type information
D2	invalid parameter; invalid data in search or command block
D3	address reference exists to deleted area
D4	command execution failure for unknown reason; PLC-3 histogram overflow
D5	data conversion error
D6	the scanner is not able to communicate with a 1771 chassis adapter
D7	the adapter is not able to communicate with the module
D8	the 1771 module response was not valid
D9	duplicated label
DA	file is open – another station owns it
DB	another station is the program owner

Syntax

PUSH *type of DF1 PLC READ command*
PUSH *remote PLC node address*
PUSH *file number of remote PLC*
PUSH *file type on remote PLC*
PUSH *starting element offset on remote PLC*
PUSH *number of elements to be transferred*
PUSH *message time-out value*
PUSH *BTR buffer and/or internal string*
PUSH **1**
PUSH *string number*
CALL **122**
POP *CALL 122 status*

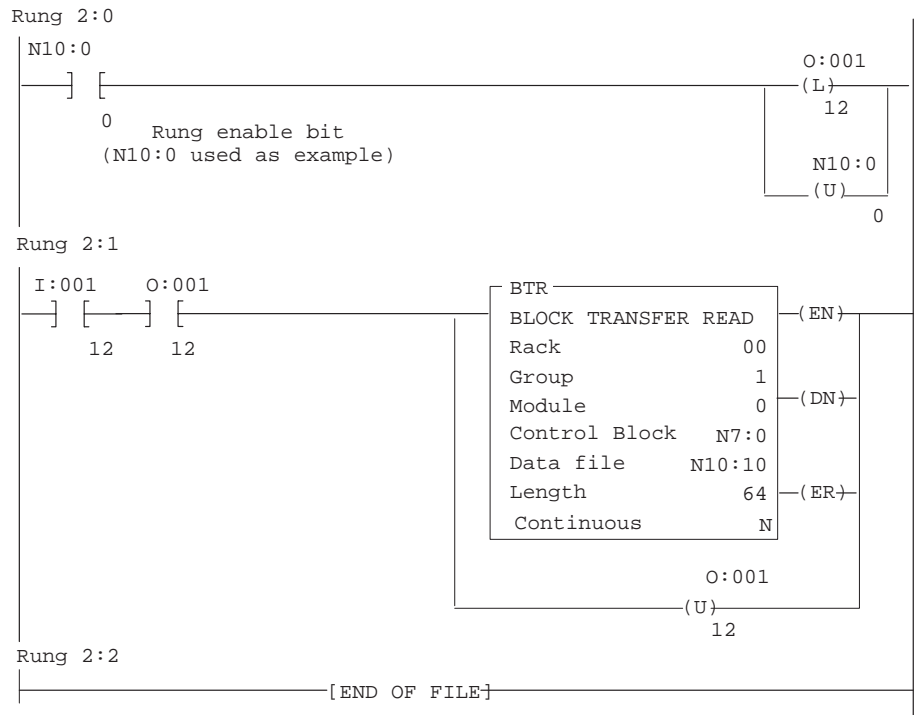
Example

```

>15  PUSH 64: CALL 4:REM SET BLOCK TRANSFER WRITE LENGTH
>16  PUSH 64: CALL 5: REM SET BLOCK TRANSFER READ LENGTH
>20  PUSH 5 : REM PLC-5 FILE
>30  PUSH 0 : REM NODE ADDRESS OF PLC-5
>40  PUSH 7 : REM FILE NUMBER OF PLC-5
>50  PUSH ASC(N) : REM FILE TYPE OF PLC-5
>60  PUSH 0 : REM STARTING WORD OFFSET OF PLC-5 FILE
>70  PUSH 20 : REM NUMBER OF DATA WORDS TO READ
>80  PUSH 10 : REM COMMAND TIME-OUT VALUE (X100MS)
>90  PUSH 1 : REM DESTINATION IS BTR BUFFER
>100 PUSH 0 : REM ALWAYS 1
>110 PUSH 0 : REM STRING NUMBER - NOT USED THIS EXAMPLE
>120 CALL 122
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 122 SETUP"

```

Sample Ladder Logic



CALL 123: Write to Remote DF1 PLC Data File

Important: This call requires the BASIC module jumper JW5 to be in 16 point mode (page 1 -7).

Use CALL 123 to write up to 63 words of data from BTW buffer and/or a string within the BASIC module to remote DF1 node (PLC-2®, -3®, or -5 processor). The table lists specific notes when using CALL 123 with the PLC-3 and PLC-5 processors.

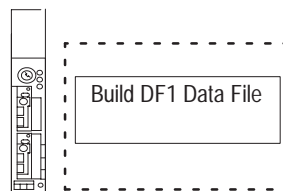
PLC processor	Notes
3	for timers and counters, the file number PUSHed (third parameter) is the structure number, limited to a maximum of 255 words input and output files cannot be accessed with this CALL. Choosing these file types will cause a 2 (bad input parameter) to be popped
5	for timer data, an element is three 16-bit words, stored in the source file in the following order: Control, Preset, and Accumulator

If you choose an internal string, the first character (transaction number) increments upon a successful write transaction to inform the BASIC module that string data was written to the PLC processor. The value of the transaction number wraps around from 255 to 0. If you perform a block-transfer the first word is reserved.

Set up the DF1 port parameters are set up with CALL 108 (page 13 -38). DF1 port can operate with either full-duplex or half-duplex slave protocol.

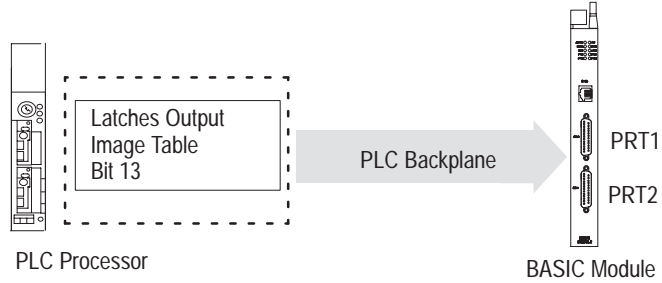
Execute CALL 123 once to set up the data transfer parameters. Input image table bit 13 and output image table bit 13 are used to initiate and notify completion of the transfer. Input image table bit 4 is used to notify the PLC that the DF1 status is available.

1. The local PLC processor ladder logic builds a file with DF1 data.

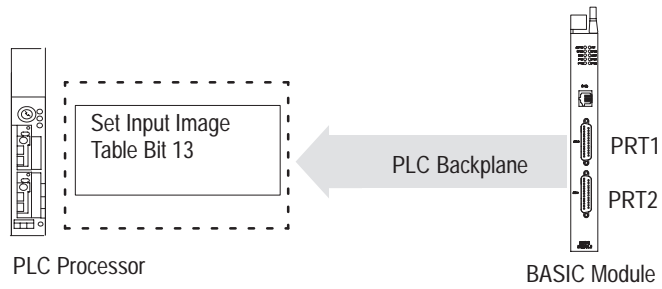


PLC Processor

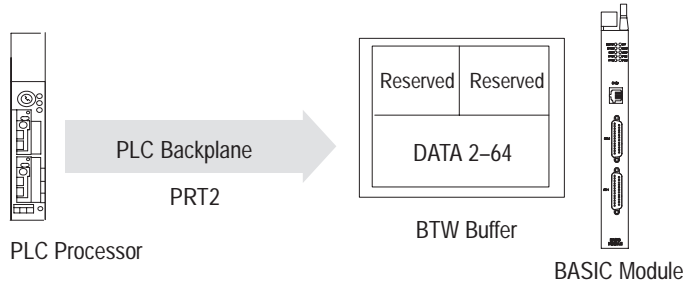
- The PLC processor latches output image table bit 13 to inform the BASIC module that valid data is available.



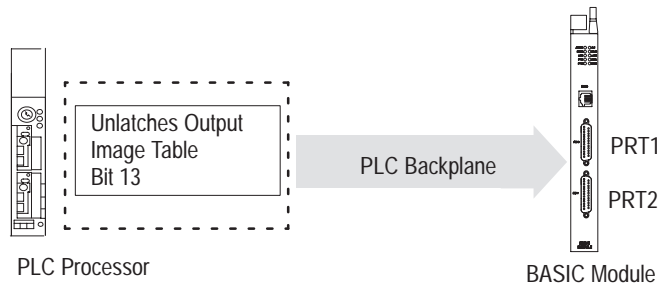
- The BASIC module sets bit 13 in the input image table to inform the PLC processor that that block transfer will be performed.



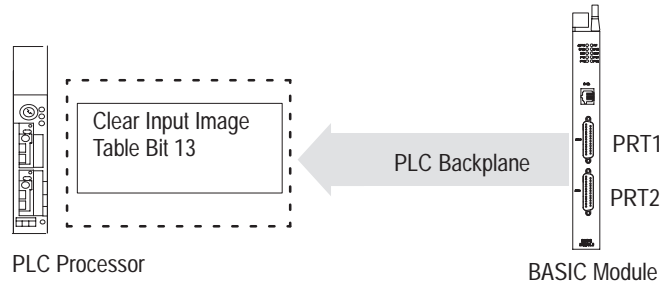
- The BASIC module performs a block transfer to receive the data.



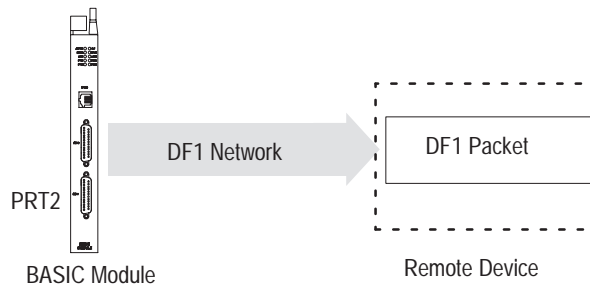
- The PLC processor unlatches bit 13 in the output image table.



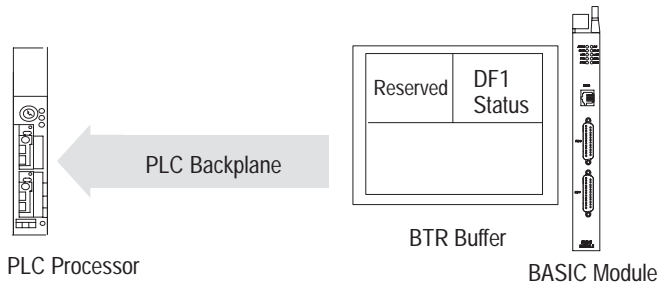
6. The BASIC module clears bit 13 in the input image table.



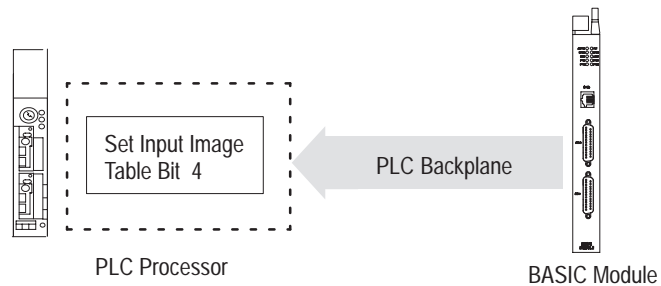
7. The BASIC module assembles the DF1 packet and sends it to the remote device.



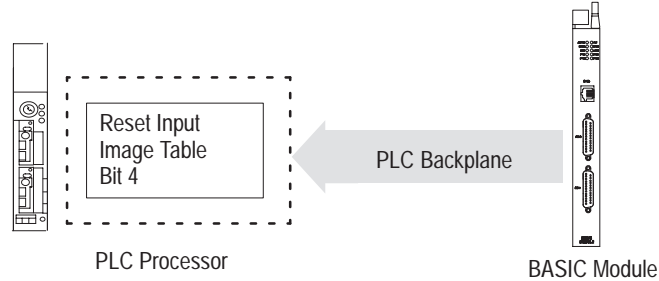
8. The BASIC module places the DF1 status of the transaction in BTR word 1.



9. The BASIC module sets the input image table bit 4 to inform the PLC processor that the data was transmitted, and that the status of the transfer is valid. The PLC processor performs a block transfer to retrieve the status.



10. The BASIC module detects a successful block transfer and resets the input image table bit 4.



This call is active until you re-execute it with different input parameters.

Input and Output Arguments

This call has ten input arguments and one output argument.

Argument	Description	Page
input 1	type of PLC WRITE command issued	13 -69
input 2	decimal node address of the remote PLC device (0–255)	13 -69
input 3	file number to be written to on PLC remote device (0–255)	13 -70
input 4	destination file time on the remote device	13 -70
input 5	starting word offset within the file	13 -70
input 6	number of elements to be transferred	13 -70
input 7	message time-out value	13 -71
input 8	selection of the BTW buffer or internal string	13 -71
input 9	always 1	13 -71
input 10	string number	13 -71
output 1	call status	13 -71

To disable this call, PUSH a zero pushed into the first input parameter. All other parameters are ignored but you must still push them.

Input Argument One

The first input argument is the type of PLC WRITE command you issued:

- 0 = disable the previously executed CALL 123
- 2 = common interface file – PLC-2 unprotected WRITE command
- 3 = PLC-3 file – word range WRITE command
- 5 = PLC-5 file – typed WRITE command

Input Argument Two

The second input argument is the decimal node address of the PLC remote device (0 through 255). If the number is not within this range, the status equals 2 and the write message does not occur.

Input Argument Three

The third input argument is the file number to be written to on the PLC remote device (0 through 255). If the number is not within this range, the status equals 2 and the write message does not occur. If you choose the common interface file in the first parameter this input is ignored, but you must still PUSH it.

Input Argument Four

The fourth input argument is the destination file type on the remote device. This number is ignored if you choose the common interface file in the first parameter (assumes integer file). If file type code is not one shown in the table the status equals 2 and write message does not take place.

File type	File type code	Words/Element (1 word = 16 bits)
integer file	ASC(N)	1 word/element
status file	ASC(S)	1 word/element
counter file	ASC(C)	3 words/element
timer file	ASC(T)	3 words/element
bit file	ASC(B)	1 word/element
control file	ASC(R)	3 words/element
input file	ASC(I)	1 word/element
output file	ASC(O)	1 word/element

Input Argument Five

The fifth input argument is the starting word offset within the file on the PLC-2 remote device (0 through 32766). For PLC-3 integer, binary, or status files, the value is 0–9999 (decimal). For PLC-3 I/O files, the value is 0–4095 (decimal). For PLC-3 timer or counter files the value is 0. If the number is not within this range, the status equals 2 and the transfer does not occur.

Input Argument Six

The sixth input argument is the number of elements to be transferred. If the number is not within the range shown below, the status equals 2 and the transfer does not occur.

File type code	Valid element length range
ASC(N)	1 to 63
ASC(S)	1 to 63
ASC(C)	1 to 21
ASC(T)	1 to 21
ASC(B)	1 to 63
ASC(R)	1 to 21
ASC(I)	1 to 63
ASC(O)	1 to 63
common interface file	1 to 63

Input Argument Seven

The seventh input argument is the message time-out value. This value (1 through 255) corresponds to the number of hundreds of milliseconds that are allowed to receive the write response (0.1 through 25.5 seconds). If the write response is not received within this time, the message aborts with the status equal to 55 in the input file word 1. If the time-out value is not within the range (1 through 255), the status equals 2 and the transfer does not take place.

Input Argument Eight

The eighth input argument is the selection of the source BTW buffer or the internal string:

- 0 = BTW buffer
- 2 = internal string

If you chose internal string (2), you can execute CALL 29 (page 12-18) to initiate each data transfer without requiring PLC processor interaction. The output table bit 13 also initiates a string transaction.

Input Argument Nine

The ninth input argument is always 1.

Input Argument Ten

The tenth input argument is the string number. If the eighth input argument does not select internal string usage, the value of this input argument is ignored, but you must still PUSH it.

Output Argument One

The output argument is the validation of the call:

- 0 = successful
- 1 = disabled
- 2 = bad input parameter
- 3 = DF1 not enabled
- 4 = string too small
- 5 = string is not dimensioned
- 6 = JW5 not in 16-point position

Whenever you attempt to write to a remote packet, the BASIC module places the status of the write into the input word 1. The possible status codes are shown in the table. This status is valid when the BASIC module sets the input table, bit 4.

Code	Indicates
0	transfer OK
1	transmission failed
2	enquiry time out
3	<ul style="list-style-type: none"> with handshaking selected – either a loss of CTS signal while transmitting or a fatal transmitter failure occurred without handshaking selected – a fatal transmitter failure occurred
4	transmission failure due to modem disconnection (DCD signal loss for more than 10 seconds) if modem handshaking with constant carrier is selected
5	DF1 driver is not enabled
6	message timed out
81	illegal command or format
82	host has a problem and will not communicate
83	remote station host is not there, disconnected, or shut down
84	host could not complete function due to hardware fault
85	addressing problem or memory protect rungs
86	function disallowed due to command protection selection
87	processor is in Program mode
88	compatibility mode file missing or communication zone problem
89	remote station cannot buffer command
8B	remote station problem due to download
8C	local station cannot execute command due to active IPBs
C1	illegal address format – field has an illegal value
C2	illegal address format – not enough fields specified
C3	illegal address format – too many fields specified
C4	illegal address format – symbol not found
C5	illegal address format – symbol is 0 or greater than the maximum number of characters supported by this device
C6	illegal address – address does not exist or does not point to something usable in this command
C7	illegal size – file is wrong size; address is past end of file
C8	cannot complete request
C9	data or file is too large
CA	request is too large; transaction size plus word address is too large
CB	access denied, privilege violation
CC	resource is not available; condition cannot be generated
CD	resource is already available; condition already exists
CE	command cannot be executed
CF	overflow; histogram overflow
D0	no access
D1	illegal data type information
D2	invalid parameter; invalid data in search or command block
D3	address reference exists to deleted area
D4	command execution failure for unknown reason; PLC-3 histogram overflow

Code	Indicates
D5	data conversion error
D6	the scanner is not able to communicate with a 1771 chassis adapter
D7	the adapter is not able to communicate with the module
D8	the 1771 module response was not valid
D9	duplicated label
DA	file is open – another station owns it
DB	another station is the program owner

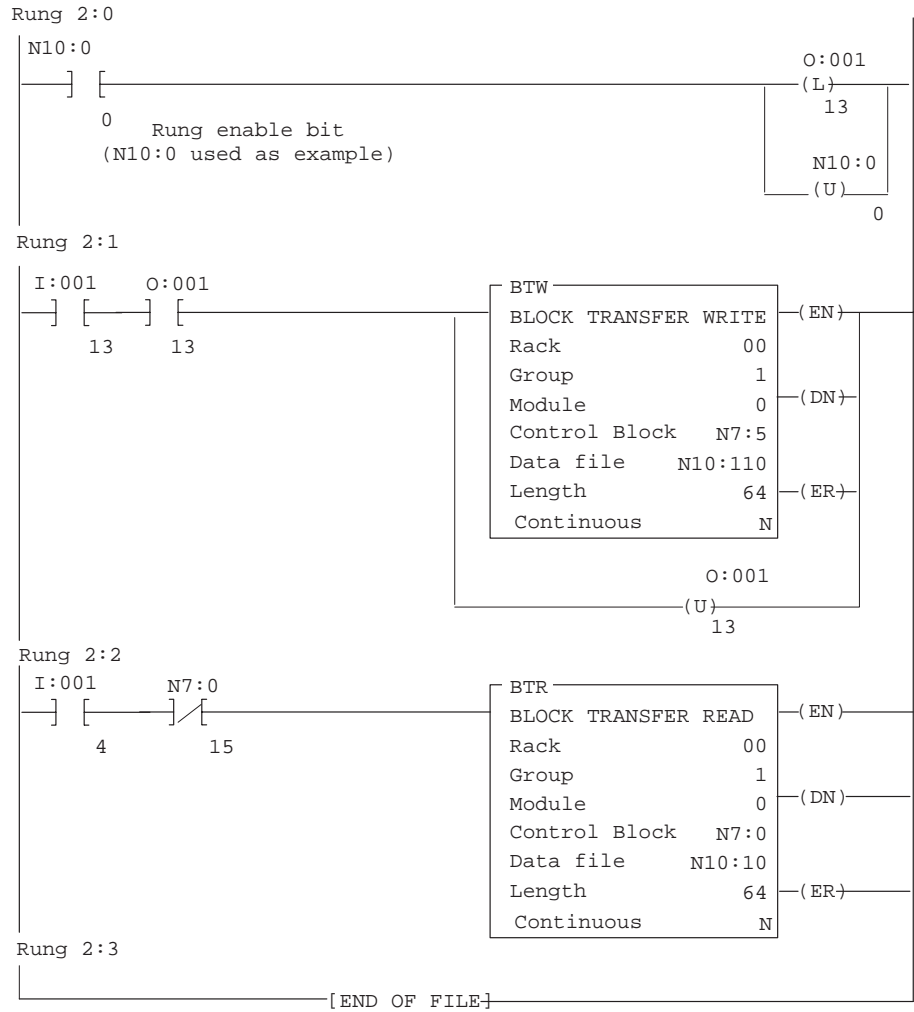
Syntax

PUSH type of PLC WRITE command
PUSH remote PLC node address
PUSH file number of remote PLC
PUSH file type on remote PLC
PUSH starting word offset on remote PLC
PUSH number of elements to be transferred
PUSH message time-out value
PUSH selection of BTW buffer or internal string
PUSH 1
PUSH string number
CALL 123
POP CALL 123 status

Example

```
>15 PUSH 64: CALL 4: REM SET BLOCK TRANSFER WRITE LENGTH
>16 PUSH 64: CALL 5: REM SET BLOCK TRANSFER READ LENGTH
>20 PUSH 5 : REM PLC-5 FILE
>30 PUSH 0 : REM PLC-5 NODE ADDRESS
>40 PUSH 7 : REM PLC-5 FILE NUMBER
>50 PUSH ASC(N) : REM PLC-5 FILE TYPE
>60 PUSH 0 : REM STARTING WORD OFFSET FOR PLC-5
>70 PUSH 20 : REM NUMBER OF WORDS TO TRANSFER
>80 PUSH 10 : REM COMMAND TIME-OUT VALUE (X100MS)
>90 PUSH 0 : REM USE BTW BUFFER
>100 PUSH 1 : REM ALWAYS 1
>110 PUSH 0 : REM STRING NUMBER-NA FOR THIS EXAMPLE
>120 CALL 123
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 123 SETUP"
```

Sample Ladder Logic



CALL 124-127

Undefined. If you execute an undefined call, you receive the error message, "ERROR-UNSUPPORTED CALL."

Product Overview

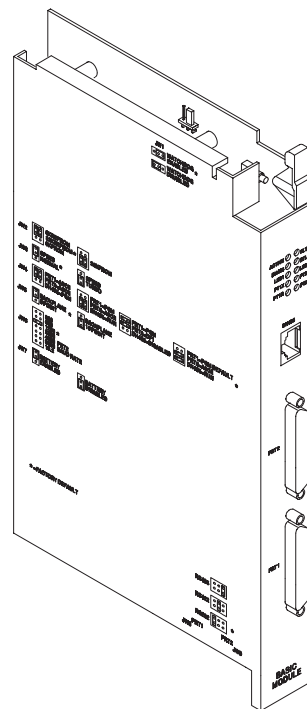
What's in This Appendix?

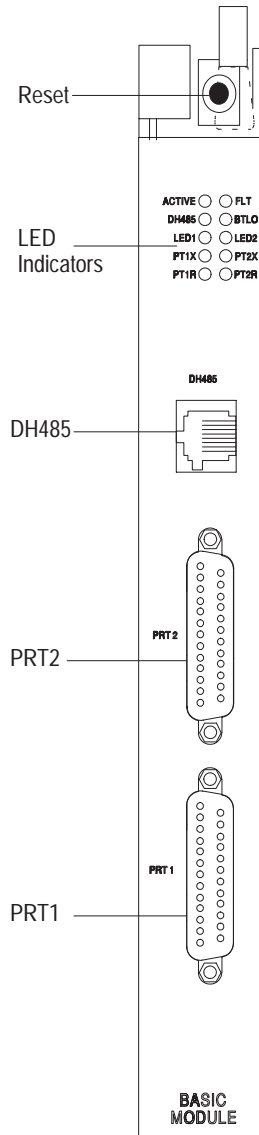
This appendix introduces you to the BASIC module.

This appendix describes:	On page:
features	A-1
programming interfaces	A-5
network configurations	A-7
memory requirements	A-10
specifications	A-11
related products	A-13

Features

The BASIC module is a single-slot module that resides in an I/O chassis. You can use the BASIC module as a foreign device interface or as an operator interface. The BASIC module provides math functions, report generation and BASIC language capabilities for any Allen-Bradley processor that communicates with the 1771 I/O system using block-transfer.



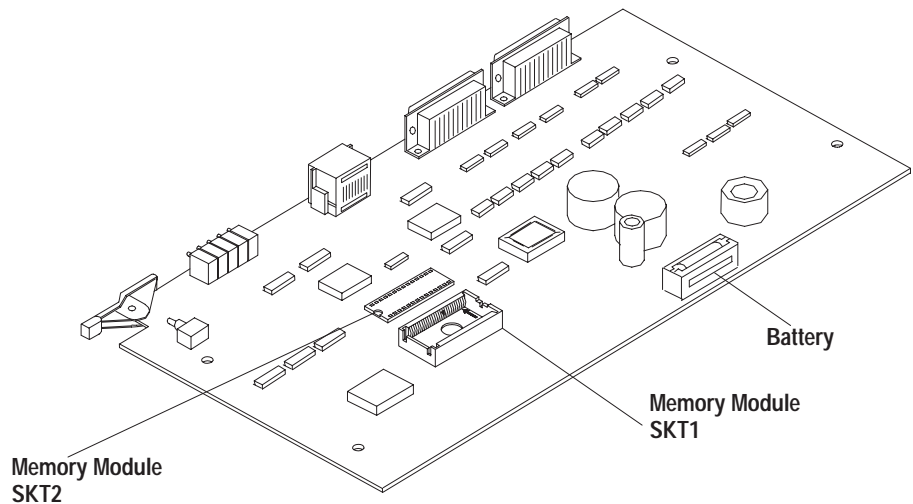


Hardware Features

Hardware element	Description
Reset switch	When you press this switch (located behind the module ejector tab), the BASIC module initiates a full reset. The BASIC module reacts to this reset the same as it does when you turn on power to your I/O chassis backplane.
LED indicators	10 LED indicators for module diagnostics and operator indicators. <ul style="list-style-type: none"> 8 pre-defined LED indicators 2 user-defined LED indicators
DH485 port	This RJ-45 port (DH485) provides communication over the DH-485 network. Use DH485 port to interface the BASIC module with the DH-485 network. This port is not isolated and cannot directly drive the DH-485 network. You must use a 1747-AIC link coupler to link port DH485 with the DH-485 network. You can also use this port as a program port.
PRT2 port	This independently configurable, isolated 25-pin D-shell serial port provides RS-232, RS-422, and RS-485 communication with I/O devices. Use PRT2 to interface the BASIC module with user devices or a modem using DF1 protocol. PRT2 provides DF1 full-duplex or half-duplex slave protocol for SCADA applications. PRT2 is capable of operating full-duplex at 300, 600, 1200, 2400, 4800, 9600, and 19200 bit/s. It is electrically isolated to 500V dc.
PRT1 port	This independently configurable, isolated 25-pin D-shell serial port provides RS-232, RS-422, and RS-485 communication with I/O devices. Use PRT1 to interface the BASIC module with user devices. PRT1 is capable of operating full-duplex at 300, 600, 1200, 2400, 4800, 9600, and 19200 bit/s. It is electrically isolated to 500V dc. You can also use this port as a program port.
RAM	24K bytes of battery backed RAM for storage of user programs and data
Battery backup	<ul style="list-style-type: none"> battery-backed, 24-hour clock/calendar capacitive backup of RAM during battery change
Memory module	<ul style="list-style-type: none"> socket for a standard EEPROM or pre-programmed EPROM memory module with a carrier socket for standard EEPROM or pre-programmed EPROM memory module without a carrier on-board EEPROM programming
User-accessible free-running clock	5 ms resolution
User-accessible wall clock/calendar	1 s resolution
Backplane interface	<ul style="list-style-type: none"> 1771 I/O supports block-transfers multiple BASIC modules can reside in the same I/O rack and function independently of each other

Software Features

Software element	Description
Programming language	Intel BASIC-52 with enhancements <ul style="list-style-type: none"> • high-level math functions • full set of trigonometric instructions • string manipulation support • floating point calculations and conversions • extensive call libraries
Block transfer communication	data read and write support with: <ul style="list-style-type: none"> • PLC-2 family processors • PLC-3 family processors • PLC-5 family processors • PLC-5/250 family processors
Program and data storage options	<ul style="list-style-type: none"> • RAM • memory modules
Communication network support	<ul style="list-style-type: none"> • DH-485 network • DF1 protocol
Data type generation	<ul style="list-style-type: none"> • 16-bit binary (4 hex digits) • SLC 16-bit signed integer • SLC 16-bit unsigned integer • 3-digit, signed, fixed decimal BCD • 4-digit, unsigned, fixed decimal BCD • 4-digit, signed, octal • 6-digit, signed, fixed decimal BCD • 3.3 digit, signed, fixed decimal BCD • 32-bit IEEE PLC-5 floating point

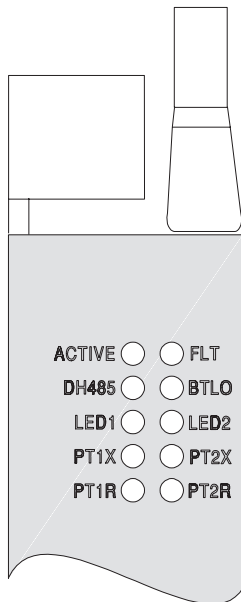
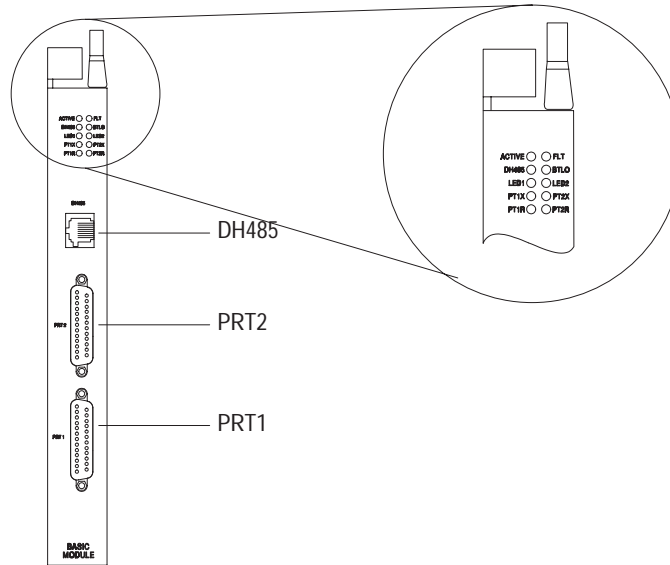


20371-M

Diagnostic Features



The BASIC module has 10 indicator LED indicators. Use these LED indicators for diagnostics and operator interface. Refer to Appendix C for more information on troubleshooting.



LED	Indication
ACTIVE	indicates the module mode and whether the BASIC module is receiving power from the backplane
FLT	indicates whether a system power problem was detected during background diagnostics
DH485	indicates whether port DH485 on the BASIC module is active for communication
BTLO	indicates whether the voltage of the battery that backs up RAM is low
LED1	user definable. LED activated through the user program.
LED2	user definable. LED activated through the user program.
PT1X	indicates whether port PRT1 on the BASIC module is transmitting signals
PT2X	indicates whether port PRT2 on the BASIC module is transmitting signals
PT1R	indicates whether port PRT1 on the BASIC module is receiving signals
PT2R	indicates whether port PRT2 on the BASIC module is receiving signals

Programming Interfaces

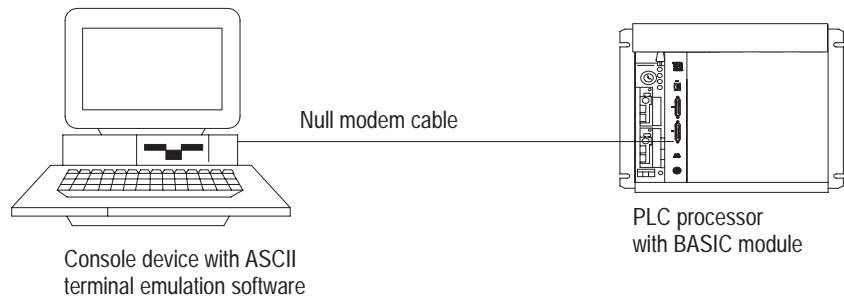
Program the BASIC module using a subset of the Intel™ BASIC 52 programming language. You can program the BASIC module using an ASCII terminal or a personal computer running ASCII terminal emulation software, such as the BASIC Development Software (catalog number 1747-PBASE).



Refer to Chapter 2 for additional information on port configuration.

ASCII Terminal Interface

Use an ASCII terminal to enter a BASIC program one line at a time to your BASIC module through port PRT1. The ASCII terminal connected to the BASIC module must be an industrial terminal, workstation, or personal computer (without the BASIC development software) that communicates in alphanumeric mode. An ASCII terminal can also be used to display charts or graphs generated by your BASIC program.



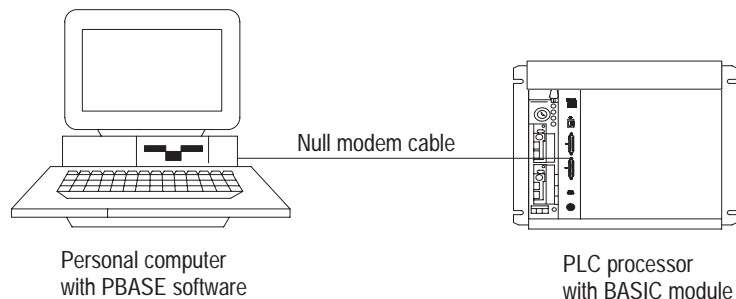
In this configuration, you connect the RS-232 port on the back of your industrial terminal or personal computer to port PRT1 on your BASIC module. Port PRT1 must be configured as the program port.

BASIC Development Software

Use a personal computer with the BASIC Development software (PBASE) to create a BASIC program that is then downloaded to your BASIC module. PBASE provides an efficient means to edit, compile (translate), upload, and download BASIC programs to the BASIC module. You can use PBASE with either the RS-232 or the DH-485 interface. You must use PBASE software when the DH485 port is the program port,

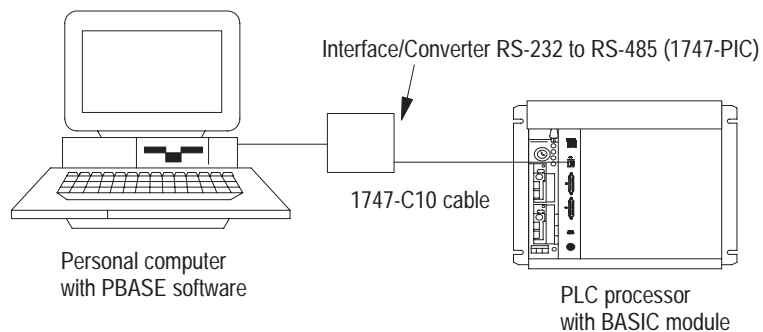
RS-232 Interface

In this configuration, you connect the serial port on the personal computer to port PRT1 or on the BASIC module. The personal computer communicates with the BASIC module through terminal emulation over an RS-232 interface. Port PRT1 is configured as the program port.



DH-485 Interface

In this configuration, you interface the serial port on the personal computer with port DH485 on the BASIC module through a 1747-PIC Interface/Converter. The 1747-PIC Interface/Converter converts the RS-232 signals from the personal computer RS-232 serial port to RS-485 format. Port DH485 is configured as the program port.



Refer to the BASIC Development Software Programming Manual (publication number 1746-6.2) for additional information on this software.

Network Configurations



Your BASIC module may communicate with a DH-485 network. It can also communicate with a remote device through a modem using the DF1 protocol. When using DF1 protocol on PRT2, port DH485 is disabled.

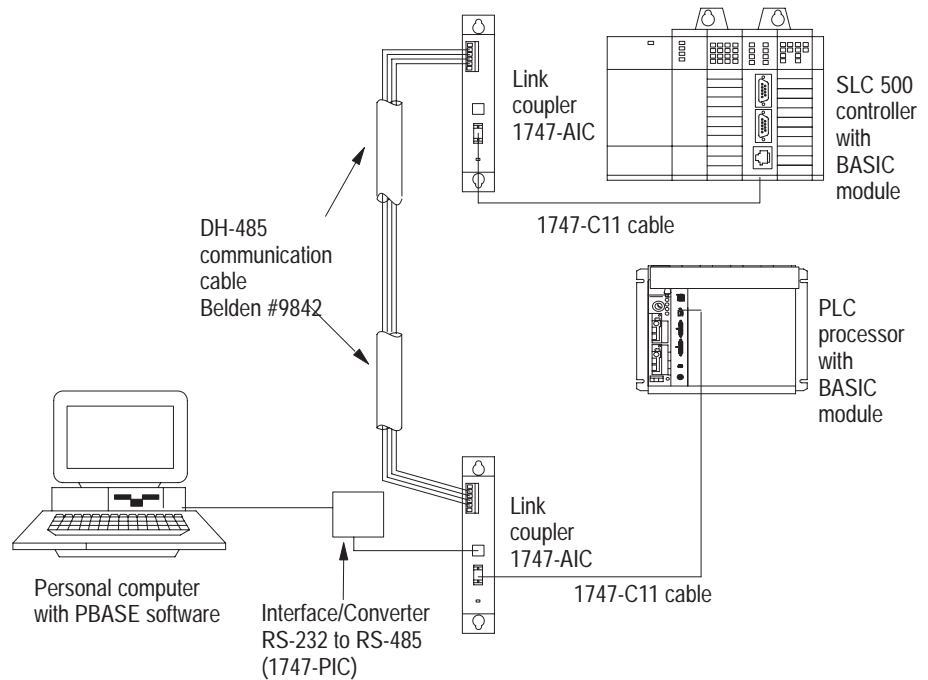
Refer to Chapter 2 for additional information on port configuration.

DH-485 Network Configuration

1747-PIC Interface/Converter/1747-AIC Link Coupler Configuration

The BASIC module interfaces with a DH-485 network through a 1747-AIC Isolated Link Coupler.

The 1747-PIC Interface/Converter converts the RS-232 signals from the personal computer RS-232 serial port to RS-485 format. The 1747-AIC link coupler links the converted signals with the DH-485 network and port DH485 on the BASIC module. The 1747-AIC link coupler also provides an interface to the DH-485 network for a personal computer with the BASIC development software. Port DH485 must be configured as the program port in order to communicate with PBASE software via the DH-485 network.

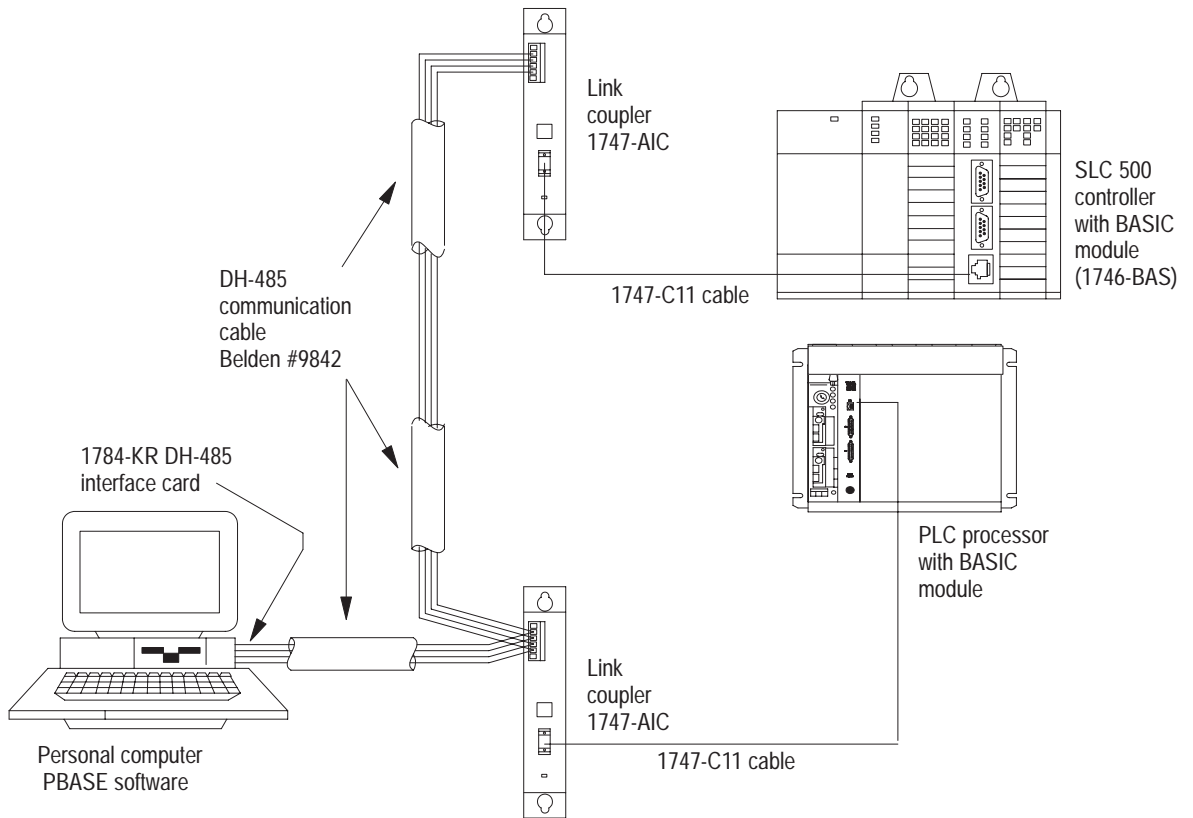


Important: Each BASIC module requires a link coupler port to interface it with the DH-485 network.

Important: When using PBASE to interface with the BASIC module, you must configure the software for DH-485 communication through the configuration and terminal selection menus.

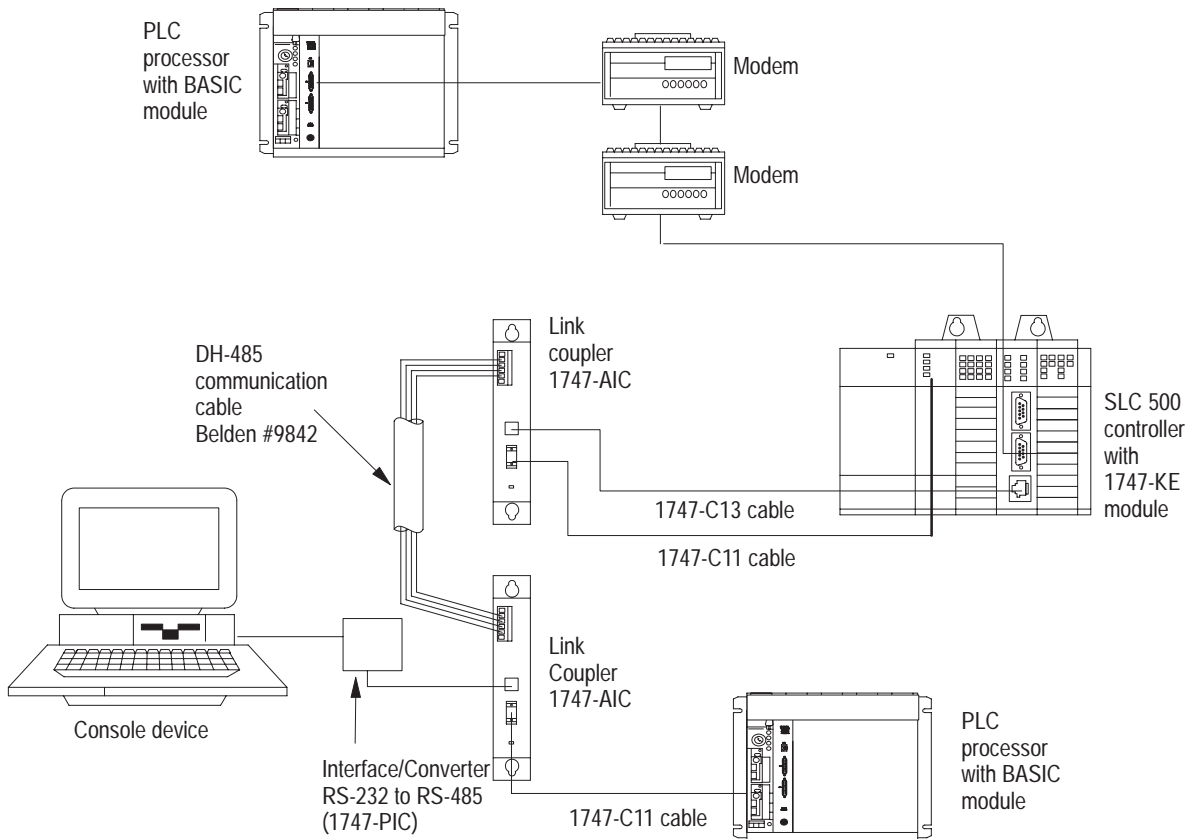
1747-AIC Link Coupler/1784-KR DH-485 Interface Card Configuration

This configuration shows the BASIC module interfaced with a DH-485 network through a 1747-AIC link coupler. The link coupler also provides an interface to the DH-485 network for a personal computer. In this configuration, a 1784-KR DH-485 Interface Card is installed in the personal computer.



DF1 Protocol Configuration

The BASIC module can use DF1 to control communications with a modem. In this configuration, the BASIC module is interfaced with a DH-485 network through a peer-to-peer communication interface with full-duplex, DF1 protocol.



Important: By configuring JW4 for DF1 communication on PRT2, DH-485 communications are disabled.



ATTENTION: Do not place the BASIC module on an active DH-485 network until the node address and communication rate of the BASIC module are configured.

Memory Requirements

The BASIC module offers two types of memory modules for BASIC programming.

- A 24K byte battery-backed RAM to store BASIC programs and protected variables
- An optional 8K or 32K byte non-volatile memory module to store BASIC programs and port configuration. You can use these memory module options with your BASIC module:
 - 8K byte EEPROM (programmable with 1771-DB/B)
 - 32K byte EEPROM (programmable with 1771-DB/B)
 - 8K byte EPROM (pre-programmed with an external PROM programmer)
 - 16K byte EPROM (pre-programmed with an external PROM programmer)
 - 32K byte EPROM (pre-programmed with an external PROM programmer)

Specifications

Environmental Conditions

Condition	Range
Operating temperature	0° C to 60° C (32° F to 140° F)
Storage temperature	-40° C to 85° C (-40° F to 185° F)
Relative humidity	5% to 95% (non-condensing)

Backplane Power Consumption

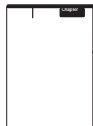
Operating voltage	Current requirement
5V dc	.75 Amps

Important: The BASIC module receives its power from the 1771-I/O backplane. The power consumption of the BASIC module must be taken into consideration when planning your PLC system. Refer to the documentation supplied with your PLC processor or 1771-I/O equipment for additional information on power supplies and current requirements.

Port Driver and Receiver

Drive output	Receiver sensitivity
+3.6 V minimum	200 mV minimum

Module Location

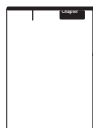


One 1771 I/O chassis module slot. See Chapter 1 for further details.

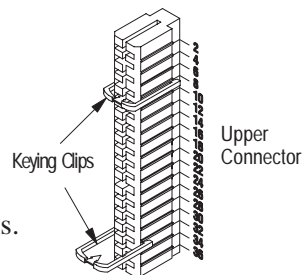
Keying

Top backplane connector:

- between 8 and 10
- between 32 and 34



See Chapter 1 for further details.



Port Isolation

Port	Isolation	Isolation voltage
PRT1	backplane to port	500V dc
PRT2	backplane to port	500V dc
PRT1 and PRT2	PRT1 to PRT2	500V dc

Important: Port DH485 is not isolated.

Clock/Calendar Accuracy

Specification	Range
accuracy	± 1 minute/month @ 25° C + 0, - 6 minute/month @ 60° C

Math

Precision	Range
8 significant digits	±1E ⁻¹²⁷ to ±.99999999E ⁺¹²⁷

Maximum Communication Distances

Communication rate (bit/s)	Maximum distance allowed meters (feet)			
	RS-232	RS-423	RS-422	RS-485
300	15 (50)	15 (50)	1230 (4000)	1230 (4000)
600	15 (50)	15 (50)	1230 (4000)	1230 (4000)
1200	15 (50)	15 (50)	1230 (4000)	1230 (4000)
4800	15 (50)	15 (50)	1230 (4000)	1230 (4000)
9600	15 (50)	15 (50)	1230 (4000)	1230 (4000)
19200	15 (50)	15 (50)	1230 (4000)	1230 (4000)

Important: Use the RS-232 jumper settings for JW8 or JW9 when communicating in RS-423 mode (page 1 -9). RS-423 devices should be unterminated and cable length should be a maximum 50 ft.

Related Products

Product	Catalog number
8K byte EEPROM memory module (supports turbo mode)	1771-DBMEM1
8K byte EEPROM memory module (supports normal mode only)	1747-M1
32K byte EEPROM memory module (supports turbo mode)	1771-DBMEM2
32K byte EEPROM memory module (supports normal mode only)	1747-M2
8K byte UVPR0M memory module (supports normal mode only)	1747-M3
32K byte UVPR0M memory module (supports normal mode only)	1747-M4
BASIC Development Software	1747-PBASE
communication cable (72" length, interchangeable with C-11 and C-20 cables)	1747-C10
communication cable (12" length, interchangeable with C-10 and C-20 cables)	1747-C11
communication cable (100" length, interchangeable with C-10 and C-11 cables)	1747-C20
communication cable (36" length, different from C-10, C-11, C-20 cables)	1747-C13
personal computer to DH-485 interface card	1784-KR
interface/converter (RS-232 to RS-485)	1747-PIC
link coupler	1747-AIC

A C tool kit is available from one of our Pyramid Solutions Program partners (see the Pyramid Solutions Program Product Directory, PSP-5.1).

For additional information on these products, refer to your local Allen-Bradley sales office.

Notes:

Conversion Table

What's in This Appendix?

The table below lists the decimal, hexadecimal, octal, and ASCII conversions.



Refer to Chapter 8 for information on data types.

Column 1				Column 2				Column 3				Column 4			
DEC	HEX	OCT	ASCII	DEC	HEX	OCT	ASCII	DEC	HEX	OCT	ASCII	DEC	HEX	OCT	ASCII
00	00	000	NUL	32	20	040	SP	64	40	100	@	96	60	140	'
01	01	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
02	02	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
03	03	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
04	04	004	EOT	36	24	044	\$	68	44	104	D	100	64	144	d
05	05	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
06	06	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
07	07	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
08	08	010	BS	40	28	050	(72	48	110	H	104	68	150	h
09	09	011	HT	41	29	051)	73	49	111	I	105	69	151	i
10	0A	012	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	VT	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	FF	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	0D	015	CR	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	0E	016	SO	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	0F	017	SI	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC	59	3B	073	;	91	5B	133	[123	7B	173	{
28	1C	034	FS	60	3C	074	<	92	5C	134	\	124	7C	174	.
29	1D	035	GS	61	3D	075	=	93	5D	135]	125	7D	175	}
30	1E	036	RS	62	3E	076	>	94	5E	135	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

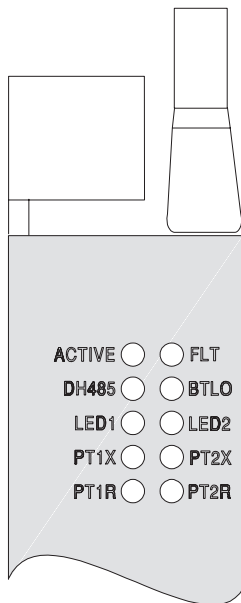
Notes:

Troubleshooting

What's in This Appendix?

This appendix describes:	On page:
interpret the indicator lights	C-1
error messages from BASIC	C-2
error messages from CALL routines	C-4

Interpret the Indicator Lights



The BASIC module has 10 indicator LED indicators.

If this LED:	Status is:	This indicates:
ACTIVE (green)	ON	the BASIC module is receiving power from the backplane and is executing BASIC code
	blinking	the BASIC module is in Command mode
	OFF	the BASIC module is not receiving power from the backplane. A fault condition exists.
FLT (red)	ON	a system problem was detected during background diagnostics. Contact your local Allen-Bradley representative.
	OFF	no system problems detected during background diagnostics
DH485 (green)	ON	port DH485 on the BASIC module is active for communication
	OFF	port DH485 on the BASIC module is not active for communication
BTLO (red)	ON	the voltage of the battery that backs up RAM is low. A new battery is needed. See Chapter 3.
	OFF	the voltage of the battery that backs up RAM is at an acceptable level
LED1 (amber)	ON	user definable. LED activated through the user program.
	OFF	user definable. LED de-activated through the user program.
LED2 (amber)	ON	user definable. LED activated through the user program.
	OFF	user definable. LED de-activated through the user program.
PT1X (green)	blinking	port PRT1 on the BASIC module is transmitting signals
	OFF	port PRT1 on the BASIC module is not transmitting signals
PT2X (green)	blinking	port PRT2 on the BASIC module is transmitting signals
	OFF	port PRT2 on the BASIC module is not transmitting signals
PT1R (green)	blinking	port PRT1 on the BASIC module is receiving signals
	OFF	port PRT1 on the BASIC module is not receiving signals
PT2R (green)	blinking	port PRT2 on the BASIC module is receiving signals
	OFF	port PRT2 on the BASIC module is not receiving signals

Error Messages from BASIC

When BASIC is in Run mode the format of the error messages is:

Error:

```

  XXX - IN LINE  YYY

  YYY BASIC STATEMENT
  -----X
  
```

Where:

XXX is the error type. The X shows 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 the command mode only the error type is printed out, not the line number.

YYY is the line number of the program in which the error occurred. For example,

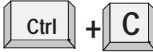
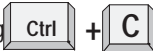
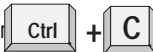
```

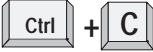
ERROR: BAD SYNTAX - IN LINE 10

10 PRINT 34*21*
-----X
  
```

Here are the BASIC module's error messages:

This error message:	Occurs when:
A-STACK	the A-stack (argument stack) pointer is forced "out of bounds." An "out of bounds" condition occurs if: <ul style="list-style-type: none"> • you overflow the argument stack by PUSHing too many expressions onto the stack, or • you attempt to POP data off the stack when no data is present
ARITH. OVERFLOW	an arithmetic operation exceeds the upper limit of a module floating point number. The largest floating point number in the BASIC Module is +/-99999999E+127. for example: 1E+70*1E+70 causes an ARITH. OVERFLOW error.
ARITH. UNDERFLOW	an arithmetic operation exceeds the lower limit of a module floating point number. The smallest floating point number in the BASIC Module is +/-1E-127. for example: 1E-80/1E+80 causes an ARITH. UNDERFLOW error.
ARRAY SIZE	an array is dimensioned by a DIM statement and you attempt to access a variable that is outside of the dimensioned bounds for example: <pre> >DIM A (10) >PRINT A(11) ERROR: ARRAY SIZE READY > </pre>

This error message:	Occurs when:
BAD ARGUMENT	<p>the argument of an operator is not within the limits of the operator</p> <p>for example:</p> <p>SQR(-12) generates a BAD ARGUMENT error because the value of the SQR argument is limited to positive numbers</p>
BAD SYNTAX	<p>an invalid BASIC module command, statement or operator is entered and BASIC cannot process the entry</p>
C-STACK	<ul style="list-style-type: none"> the C-stack (control stack) pointer is forced "out of bounds" you attempt to use more control stack than is available in the module you execute a RETURN before a GOSUB, a WHILE or UNTIL before a DO, or a NEXT before a FOR (See Chapter 11 for a description of these statements.) you jump out of loop structures such as DO WHILE
CANT CONTINUE	<ul style="list-style-type: none"> you edit the program after halting execution and then enter the CONT command you enter  while in a call routine <p>you can halt program execution by either entering  or by executing a STOP statement. Normally, program execution continues after entering the CONT command. You must enter  during program execution or you must execute a STOP statement before the CONT command can work.</p>
DIVIDE BY ZERO	<p>you attempt to divide by zero</p>
EXTRA IGNORED	<p>an INPUT statement (page 11 -17) requiring numeric data, receives numeric data followed by letters. Letters are ignored. Error also occurs from CALL 61 (page 12 -60).</p>
MEMORY ALLOCATION	<ul style="list-style-type: none"> user memory (RAM) is full BASIC cannot determine memory bounds because the system control value MTOP is altered RAM contains an incomplete program file you attempt to access STRINGS that are "outside" the defined string limits
NO DATA	<p>a READ statement is executed and no DATA statement exists or all DATA was read and a RESTORE instruction was not executed. (See Chapter 11 for a description of these statements.)</p> <p>the message ERROR: NO DATA - IN LINE XXX is printed to the console device.</p>
PROGRAMMING	<p>the BASIC module is programming an EEPROM. An error during programming destroys the EPROM file structure. You cannot save any more programs on that particular EEPROM once a PROGRAMMING error occurs. If the EEPROM size is exceeded, the previously stored program may be partially altered.</p>

Application errors such as divide by zero error, syntax error, receipt of a  (page 10 -4), and execution of STOP (page 11 -36) or END (page 11 -10) statements cause the BASIC module to return to the Command mode from Run mode. Use CALL 38 (EXPANDED ONERR) (page 12 -36) to jump to an interrupt routine instead of returning to the Command mode.

Error Messages from CALL Routines

Your module generates these messages if an error occurs while the module tries to execute a CALL routine.

PRT2 Port Support CALL Error Messages

This error message:	Occurs when:
INVALID INPUT DATA	you enter an invalid value when using CALL 30
INVALID VALUE PUSHED	you enter a value other than 0, 1 or 2 when using CALL 37

Wall Clock CALL Error Messages

This error message:	Occurs when:
INSUFFICIENT NUMBER OF STRING CHARACTERS ALLOCATED	you attempt to execute a CALL 43, 45 or 52 and a string length of 18, 8 or 9, respectively, is not allocated during string allocation
INVALID DATE/TIME PUSHED	you enter an invalid value for the date and/or time when using CALL 40 and 41
NUMBER BYTES/STRING EXCEED 254	using CALL 43, 45 or 52 and the STRING X,X command allocates more characters per string than is allowed For example: 10 STRING 1000,300
INVALID NUMBER PUSHED	you push an invalid string value or day of week value using CALL 42, 43 and 52.
STRING # NOT ALLOCATED	you attempt to access a string that is outside the allocated string memory when using CALLs 60, 61, 64, 65, 66, 67 or 68 for example: >10 STRING 100,9 >20 PUSH 5, 12 >30 CALL 60 >RUN ERROR - STRING # NOT ALLOCATED error occurs because STRING 12 is outside the area reserved for strings.
# BYTES/STRING EXCEED 254	the STRING X,X command allocates more characters per string than is allowed using CALL 62 for example: >10 STRING 1000,300
INSUFFICIENT NUMBER OF STRING CHARACTERS	you do not use the required minimum string lengths when using CALL 62
BAD # PUSHED	the string position pointer is zero (invalid position) using CALL 66

String Support CALL Error Messages

This error message:	Occurs when:
INSUFFICIENT STRING SIZE	<p>the resulting string cannot hold all required characters when using CALLS 61 or 66</p> <p>for example:</p> <pre>>10 STRING 100,9 REM MAX OF 9 CHR'S/STRING >20 \$(0)="01234567" >30 \$(1) = "890"</pre> <p>if you attempt to insert or concatenate, an error occurs because the resulting string requires 11 characters</p>
BAD POSITION	<p>you attempt to access a string position that is beyond the declared length of the string when using CALL 66</p> <p>for example:</p> <pre>>10 STRING 110,9 >20 \$(0)="1234" >30 \$(1)="56" >40 PUSH 6 REM INVALID POSITION OF \$(0) >50 PUSH 1 REM \$(1) >60 PUSH 0 REM \$(0) >70 CALL 66 REM INSERT \$(1) INTO \$(0) @ POS 6</pre> <p>error occurs because position 6 is outside of the declared string.</p>
EXTRA IGNORED	<p>the resulting string cannot hold all the characters when using CALL 61. Similar to INSUFFICIENT STRING SIZE error. Extra characters are lost.</p>
INPUT	<p>an input statement requiring numeric data, received numeric data followed by letters. The letters are ignored.</p> <p>for example:</p> <pre>INPUT A</pre> <p>entering 1.23AB causes this error message. The program continues to run.</p>

Memory Support CALL Error Messages

This error message:	Occurs when:
INVALID MTOP ADDRESS ENTERED	<p>you select an invalid RAM location for a new MTOP value when using CALL 77</p>
PROGRAM NOT FOUND	<p>a program number higher than 255 is PUSHed when using CALL 71 or 72 or if the program is not found</p>

Miscellaneous CALL Error Messages

This error message:	Occurs when:
INVALID BAUD RATE ENTERED	a communication rate other than 300, 600, 1200, 2400, 4800, 9600 or 19200 bit/s is PUSHed using CALL 78
INCOMPLETE ROM PROGRAM FOUND	CALL 81 detects an incomplete program in the memory module. You can burn no additional programs onto this EEPROM. All earlier programs are still accessible
NO PROGRAM FOUND BUT THE PROM IS NOT BLANK	miscellaneous data is found on a "blank" PROM using CALL 81. You should clear the PROM before using it.

Series A Configuration Plugs

What's in This Appendix?

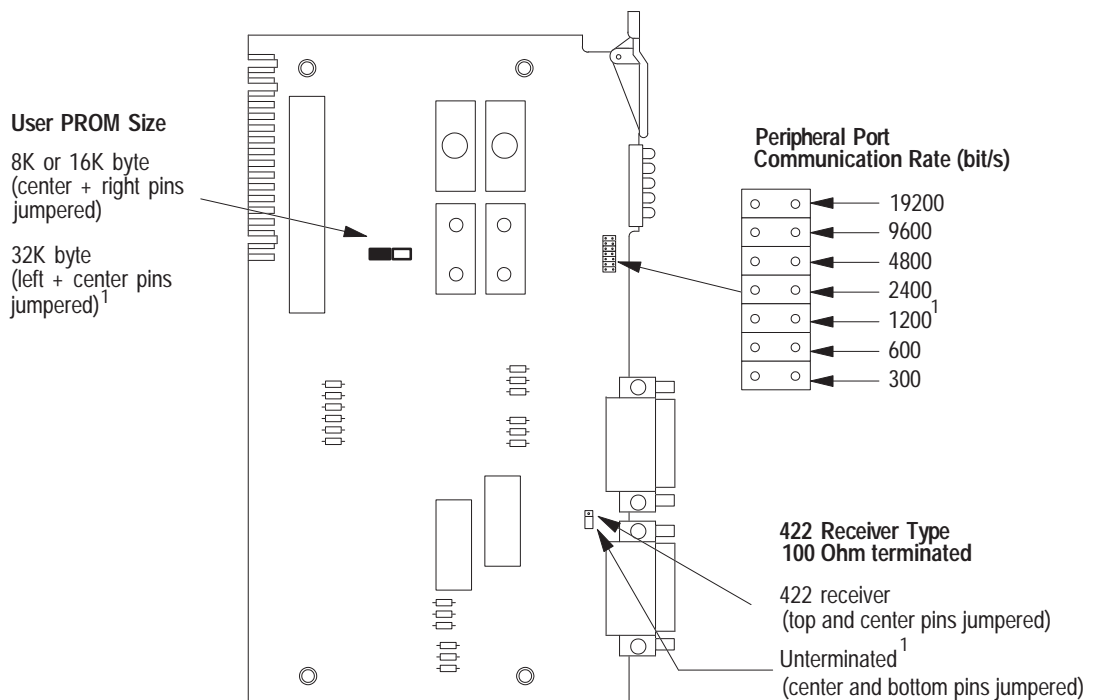


This appendix gives you the configuration plug settings for the **Series A BASIC** module. If you have a Series B module this information is not relevant. The Series B module does not have configuration plugs. However, it does have configuration jumpers. These jumper settings are described in Chapter 1.

Configuration Plugs

There are three sets of user selectable configuration plugs on the Series A BASIC module. You can use these configuration plugs to select:

- EPROM size
- peripheral port communication rate (bit/s)
- 422 receiver termination



¹Factory Setting

17898

Important: All other configuration plugs are factory set. Do not reset the other factory set configuration plugs.

Notes:

Quick Reference

What's in This Appendix?



This appendix gives you an alphabetical quick reference table to the different, operators (Chapter 9), commands (Chapter 10), statements (Chapter 11), and calls (Chapters 12 and 13) the BASIC module supports.

BASIC	Description	Page
ABS	return the absolute value of expression	9 -11
ADD (+)	add expressions together	9 -5
.AND.	combine the first expression with the second expression using .AND.	9 -7
ASC	return integer value of ASCII character	9 -14
ATN	return arctangent of argument	9 -11
BRKPNT	sets a breakpoint	10 -2
CALL 0	reset module	12 -2
CALL 2	timed-block-transfer-read buffer	12 -2
CALL 3	timed-block-transfer-write buffer	12 -3
CALL 4	set block-transfer-write length	12 -4
CALL 5	set block-transfer-read length	12 -4
CALL 6	block-transfer-write buffer	12 -5
CALL 7	block-transfer-read buffer	12 -5
CALL 10	3-digit decimal BCD to BASIC floating point	12 -6
CALL 11	16-bit binary to BASIC floating point	12 -7
CALL 12	4-digit octal to BASIC floating point	12 -7
CALL 13	6-digit decimal BCD to BASIC floating point	12 -8
CALL 14	SLC 16-bit signed integer to BASIC floating point	12 -8
CALL 15	SLC 16-bit unsigned integer to BASIC floating point	12 -9
CALL 16	enable/disable DF1 packet interrupt	12 -10
CALL 17	4-digit BCD to BASIC floating point	12 -11
CALL 18	re-enable control C break function	12 -11
CALL 19	disable the control C break function	12 -12
CALL 20	BASIC floating point to 3-digit decimal BCD	12 -12
CALL 21	BASIC floating point to 16-bit binary	12 -13
CALL 22	BASIC floating point to 4-digit octal	12 -13
CALL 23	BASIC floating point to 6-digit decimal BCD	12 -14
CALL 24	BASIC floating point to SLC 16-bit signed integer	12 -15
CALL 25	BASIC floating point to SLC 16-bit binary	12 -16
CALL 26	BASIC floating point to 3.3-digit BCD	12 -17
CALL 27	BASIC floating point to 4-digit BCD	12 -17
CALL 29	read/write to PLC/SLC from module internal string	12 -18
CALL 30	PRT2 port support parameter set	12 -20

BASIC	Description	Page
CALL 31	display PRT2 port parameters	12 -21
CALL 32	enable/disable processor interrupt	12 -22
CALL 33	transfer data from PRT1/PRT2 to BTR buffer	12 -23
CALL 34	transfer data from BTW buffer to PRT1/PRT2	12 -29
CALL 35	retrieve numeric input character from PRT2 port	12 -34
CALL 36	get number of characters in PRT2 port buffers	12 -35
CALL 37	clear PRT2 port buffers	12 -35
CALL 38	expanded ONERR restart	12 -36
CALL 39	3.3-Digit Signed, BCD to BASIC Floating Point	12 -38
CALL 40	set wall clock time	12 -39
CALL 41	set wall clock date	12 -40
CALL 42	set wall clock day of week	12 -40
CALL 43	date/time retrieve string	12 -41
CALL 44	date retrieve numeric	12 -41
CALL 45	time retrieve string	12 -42
CALL 46	time retrieve numeric	12 -42
CALL 47	retrieve day of week string	12 -43
CALL 48	retrieve day of week numeric	12 -43
CALL 49	read remote DH-485 SLC data file	12 -44
CALL 50	write to remote DH-485 SLC data file	12 -50
CALL 52	date retrieve string	12 -58
CALL 60	string repeat	12 -59
CALL 61	string append	12 -60
CALL 62	number to string conversion	12 -61
CALL 63	string to number conversion	12 -62
CALL 64	find a string in a string	12 -63
CALL 65	replace a string in a string	12 -64
CALL 66	insert a string in a string	12 -65
CALL 67	delete a string from a string	12 -66
CALL 68	determine length of a string	12 -67
CALL 70	ROM to RAM program transfer	13 -2
CALL 71	ROM/RAM to ROM program transfer	13 -3
CALL 72	RAM/ROM return	13 -4
CALL 73	battery-backed RAM disable	13 -5
CALL 74	battery-backed RAM enable	13 -5
CALL 77	protected variable storage	13 -6
CALL 78	set program port communication rate	13 -8
CALL 79	set active LED blinking state (no operation)	13 -8
CALL 80	check battery condition	13 -9
CALL 81	user PROM check and description	13 -10
CALL 82	check user memory module map	13 -11
CALL 83	display DH485 port setup	13 -11
CALL 84	transfer DH-485 CIF to BASIC input buffer	13 -12
CALL 85	transfer BASIC output buffer to DH-485 CIF file	13 -13
CALL 86	check DH-485 interface remote write status	13 -14

BASIC	Description	Page
CALL 87	check DH-485 interface file remote read status	13 -15
CALL 88	BASIC floating point to PLC-5 floating point	13 -16
CALL 89	PLC-5 floating point to BASIC floating point	13 -17
CALL 90	read remote DH-485 data file to BASIC input buffer	13 -18
CALL 91	write BASIC output buffer to remote DH-485 data file	13 -22
CALL 92	read remote DH-485 CIF to BASIC input buffer	13 -26
CALL 93	write output buffer to remote DH-485 CIF file	13 -29
CALL 94	display current PRT1 port setup	13 -32
CALL 95	get number of characters in PRT1 buffers	13 -32
CALL 96	clear PRT1 receive/transmit buffers	13 -33
CALL 97	enable PRT2 DTR signal	13 -33
CALL 98	disable PRT2 DTR signal	13 -34
CALL 99	reset print head pointer	13 -34
CALL 100	download/program assembly language to EEPROM	13 -35
CALL 101	upload user (E)EPROM code to host	13 -35
CALL 103	print PRT1 transmit buffer and pointer	13 -36
CALL 104	print PRT1 receive buffer and pointer	13 -37
CALL 105	reset PRT1 to default settings	13 -37
CALL 108	enable DF1 driver communications	13 -38
CALL 109	print the argument stack	13 -44
CALL 110	print the PRT2 port transmit buffer and pointer	13 -45
CALL 111	print the PRT2 receive buffer and pointer	13 -46
CALL 112	user LED control	13 -47
CALL 113	disable DF1 driver communications	13 -47
CALL 114	transmit DF1 packet	13 -48
CALL 115	check DF1 status	13 -49
CALL 116	call user defined assembly language routine	13 -50
CALL 117	get DF1 packet length	13 -51
CALL 118	PLC/SLC unsolicited writes	13 -52
CALL 119	reset PRT2 port to default settings	13 -56
CALL 120	clear BASIC module I/O buffers	13 -57
CALL 122	read remote DF1 PLC data file	13 -58
CALL 123	write to remote DF1 PLC data file	13 -66
CBY	retrieve data from core or program memory address location	9 -18
CHR	convert numeric expression to ASCII value	9 -16
CLEAR	sets all variables equal 0 and resets all BASIC evoked interrupts and stacks	11 -2
CLEARI	resets all BASIC evoked interrupts	11 -3
CLEAR S	resets all stacks	11 -3
CLOCK0	disable or turn off free-running clock	11 -4
CLOCK1	enable or turn on free-running clock	11 -5
CONT	continue after a CTRL-C	10 -3
COS	return the cosine of argument	9 -10
CTRL C	stops program execution	10 -4

BASIC	Description	Page
CTRL Q	restart a LIST or PRINT interrupted by CTRL-S	10 -5
CTRL S	interrupt the scrolling of code during a LIST or PRINT	10 -6
DATA	specify the expressions that you can retrieve with a READ	11 -6
DBY	retrieve/assign data to/from internal data memory	9 -18
DIM	reserve storage for arrays	11 -7
DIVIDE (/)	divide first expression by second expression	9 -5
DO-UNTIL	set up loop control	11 -8
DO-WHILE	set up loop control	11 -9
EDIT	access the BASIC line editor	10 -7
END	terminate program execution	11 -10
EOF	test for empty input buffer	9 -17
ERASE	delete BASIC program stored in EEPROM through a PROG	10 -8
EXP	raise e to power of argument	9 -13
**	raise first expression by the power of the second expression	9 -5
FREE	list available bytes in RAM	9 -17
FOR-TO-(STEP)-NEXT	set up loop control	11 -11
GET	read console input device	11 -12
GOSUB	transfer control to a subroutine	11 -13
GOTO	transfer control to line specified	11 -14
IDLE	forces module to wait for an interrupt	11 -14
IF-THEN-ELSE	set up a conditional test	11 -15
INPL	read entire line from program port buffer	11 -16
INPS	read an entire string from program port buffer	11 -16
INPUT	enter data from consoled device	11 -17
INT	return integer portion of expression	9 -12
LEN	list amount of bytes in current program	9 -17
LOG ()	return the natural log of the argument	9 -13
LD@	retrieve floating point numbers stored with ST@	11 -18
LET	assign a variable to the value of an expression	11 -19
LIST	list the program code	10 -9
MODE	set port parameters for PRT1, PRT2, and DH485	11 -20
MTOP	return last valid memory address	9 -18
MULTIPLY (*)	multiply expressions together	9 -5
-	negation	9 -6
NEW	delete program and all variables currently stored in RAM	10 -10
NEXT	returns FOR-TO-NEXT loop to beginning of loop	11 -21
NOT	returns the one's complement or inverse of the number	9 -11
NULL	set number of null characters to output after a carriage return in a print statement	10 -10
ONDF1	enable or disable the DF1 packet interrupt capability	11 -22
ONERR	handle arithmetic errors during program execution	11 -23

BASIC	Description	Page
ON-GOSUB	transfer control to subroutine when expression following ON is encountered	11 -24
ON-GOTO	transfer control to line specified when expression following ON is encountered	11 -26
ONTIME	use to compensate for incompatibility between timers/counters on the microprocessor and the BASIC module	11 -25
.OR.	combine the first expression with the second expression using .OR.	9 -8
PI	store constant. PI	9 -12
PH0. and PH1.	direct BASIC module to print a number in hexadecimal format to a console device	11 -27
POP	remove value from argument stack	11 -28
PRINT	output value to the console device	11 -29
PROG	program resident EEPROM with current program in RAM	10 -11
PROG1	program resident EEPROM with port information for all three ports as well as MTOP information	10 -12
PROG2	program resident EEPROM with port information for all three ports as well as MTOP information and execute first program in the EEPROM	10 -13
PUSH	place expression on argument stack	11 -30
RAM	select current program out of RAM	10 -15
READ	retrieve expressions specified in DATA statement	11 -31
REM	specify a comment line in BASIC program	11 -32
REN	renumber program lines	10 -16
RESTORE	reset internal pointer to beginning of data to read it again	11 -32
RETI	exit from an interrupt that is processed	11 -33
RETURN	return control back to program after executing a GOSUB	11 -34
RND	return a pseudo-random number	9 -12
ROM	select current program out of EEPROM or EPROM	10 -17
RROM	select and run current program out of EEPROM or EPROM	10 -18
RUN	set variables to zero, clear BASIC evoked interrupts, and begin program execution with the first line number of selected program	10 -19
SGN	return the sign of argument	9 -12
SIN	return the sine of argument	9 -10
SQR	return the square root of the argument	9 -12
SINGLSTP	initiate single-step execution	10 -20
ST@	store floating point numbers to a specified address	11 -35
STOP	break program execution at specific points in the program	11 -36
STRING	allocate memory for strings	11 -37
SUBTRACT (-)	subtract one expression from another	9 -5
TAN	return the tangent of argument	9 -10
TIME	read/assign the free running clock	9 -19

BASIC	Description	Page
VER	print current version of the firmware	10 -21
XBY	read/assign external data memory	9 -19
XFER	transfer the current selected program in ROM to RAM and select RAM mode	10 -21
.XOR.	combine the first expression with the second expression using .XOR.	9 -8
@ or #	communication direction	9 -17
=	allow the first expression to equal the second expression	9 -9
<	allow the first expression to be less than the second expression	9 -9
<=	allow the first expression to be less than or equal to the second expression	9 -9
>	allow the first expression to be greater than the second expression	9 -9
>=	allow the first expression to be greater than or equal to the second expression	9 -9
<>	allows the first expression to be unequal to the second expression	9 -9

Symbols

.AND., logical operator, 9-7
 .OR., logical operator, 9-8
 .XOR., logical operator, 9-8
 # BYTES/STRING EXCEED 254, C-4
 # operator, SOC-3, 9-17
 @ operator, SOC-3, 9-17
 +, addition operator, 9-5
 -
 negation operator, 9-6
 subtraction operator, 9-5
 *, multiplication operator, 9-5
 **, exponentiation operator, 9-5
 See also EXP
 /, division operator, 9-5
 =, equal to relational operator, 9-9
 <, less than relational operator, 9-9
 <=, less than or equal to relational operator, 9-9
 >, greater than relational operator, 9-9
 >=, greater than or equal to relational operator, 9-9
 <>, not equal to relational operator, 9-9

Numbers

1-slot addressing, 1-9
 1/2-slot addressing, 1-9
 16 point mode, SOC-2, 5-2
 See also data tables; JW5
 CALL 118, 13-52
 CALL 122, 13-58
 CALL 123, 13-66
 CALL 32, 12-22
 CALL 33, 12-23
 CALL 34, 12-29
 CALL 49, 12-44
 CALL 50, 12-50
 configuration jumper JW5, 1-6
 16-bit binary, 8-5
 See also CALLs 11 and 25

16-bit binary to BASIC floating point, CALL 11, 12-7
 See also CALL 21
 1747-M1, 8K EEPROM, 3-4
 1747-PIC Interface/Converter, 2-12, A-7
 1747-AIC Isolated Link Coupler, 2-11, 2-12, A-7
 1747-C10 Cable, 2-10, 2-11
 1747-C11 Cable, 2-10, 2-11
 1747-C13 Cable, 2-10, 2-11
 1747-C20 Cable, 2-10, 2-11
 1747-KE DH-485/RS-232C Communication Interface Module, 2-10, 2-11, A-8
 1747-PIC Interface/Converter, 2-9, 2-11, 2-12, A-7
 1747-M2, 32K EEPROM, 3-4
 1747-M3, 8K EPROM, 3-4
 1747-M4, 32K EPROM, 3-4
 1770-SA/SB data recorder, SOC-5
 1770-KF3 DH-485 Communication Interface Module, 2-10, 2-11, A-8
 1771-DBMEM1, 8K EEPROM, 3-4
 1771-DBMEM2, 32K EEPROM, 3-4
 1784-KR DH-485 Interface Card, 2-11, 2-13, A-7
 2-slot addressing, 1-9
 3.3-digit, signed, fixed decimal BCD, 8-8
 See also CALL 26, 39
 3.3-digit signed, fixed decimal BCD to BASIC floating point, CALL 39, 12-38
 See also CALL 26
 3-digit signed BCD to floating point, CALL 10, 12-6
 See also CALL 20
 3-digit, signed, fixed decimal BCD, 8-5
 See also CALLs 10 and 20

- 4-digit BCD to BASIC floating point, CALL 17, 12-11
 - See also* CALL 27
 - 4-digit signed octal to BASIC floating point, CALL 12, 12-7
 - See also* CALL 22
 - 4-digit, signed octal, 8-6
 - See also* CALLs 12 and 22
 - 4-digit, unsigned, fixed, decimal BCD, 8-6
 - See also* CALLs 17 and 27
 - 6-digit signed, fixed decimal BCD to BASIC floating point, CALL 13, 12-8
 - See also* CALL 23
 - 6-digit, signed, fixed decimal BCD, 8-7
 - See also* CALLs 13 and 23
 - 8 point mode, SOC-2, 5-2
 - See also* data tables; JW5
 - configuration jumper JW5, 1-6
- A**
- A-stack, 8-1, 11-28, C-2
 - See also* argument stack
 - print, CALL 109, 13-44
 - abbreviations, Using-3
 - ABS, functional operator, 9-11
 - add (+), arithmetic operator, 9-5
 - addressing, 1-9, 5-3
 - 1-slot, 1-9
 - 1/2-slot, 1-9
 - 2-slot, 1-9
 - Allen-Bradley support, Using-6
 - AND, 9-7
 - argument stack, 8-1, 11-28, C-2
 - See also* A-stack
 - print, CALL 109, 13-44
 - ARITH. OVERFLOW, C-2
 - ARITH. UNDERFLOW, C-2
 - arithmetic operators, 9-5
 - add(+), 9-5
 - divide(/), 9-5
 - exponentiation(**), 9-5
 - See also* EXP
 - multiply(*), 9-5
 - negation(-), 9-6
 - overflow and division by zero, 9-6
 - See also* ONERR, CALL 38
 - subtract(-), 9-5
 - ARRAY SIZE, C-2
 - arrayed variables, 9-2
 - ASC, string operator, 9-14
 - See also* STRING
 - ASCII conversion table, B-1
 - ASCII port
 - See also* PRT1 port, PRT2 port
 - configuration jumper JW4, 1-5
 - definition, Using-3
 - parameters, 2-8
 - See also* MODE, CALL 30
 - PRT1 port, 2-8
 - PRT2 port, 2-8
 - ASCII terminal emulator, 4-3
 - ASCII terminal interface, A-5
 - assembly language
 - call routine, CALL 116, 13-50
 - download/program EEPROM, 13-35
 - asynchronous block-transfer, PLC-5 processor, 5-10
 - asynchronous block-transfers, 5-3
 - ATN, trigonometric operator, 9-11
 - audience of manual, Using-1
- B**
- background operations calls, 7-6
 - backplane configuration, JW5, 1-6
 - backplane conversion calls, 7-4
 - See also* Chapter 8
 - backplane conversion data, 8-4
 - backplane conversion table, B-1
 - backplane interface, A-2
 - BAD # PUSHED, C-4

- BAD ARGUMENT, C-3
- BAD POSITION, C-5
- BAD SYNTAX, C-3
- BASIC, definition, Using-3
- BASIC Development Software, ordering, A-13
- BASIC Development Software (PBASE), Using-3, A-6
 - RS-232 interface, 2-9
 - RS-485 interface, 2-9
- BASIC floating point 3.3-digit signed BCD, CALL 26, 12-17
 - See also* CALL 39
- BASIC floating point to 16-bit binary, CALL 21, 12-13
 - See also* CALL 11
- BASIC floating point to 3-digit, signed, fixed decimal BCD, CALL 20, 12-12
 - See also* CALL 10
- BASIC floating point to 4-digit BCD, CALL 27, 12-17
 - See also* CALL 17
- BASIC floating point to 4-digit signed octal, CALL 22, 12-13
 - See also* CALL 12
- BASIC floating point to 6-digit signed fixed decimal BCD, CALL 23, 12-14
 - See also* CALL 13
- BASIC floating point to PLC-5 floating point, CALL 88, 13-16
 - See also* CALL 89
- BASIC floating point to SLC 16-bit binary, CALL 25, 12-16
 - See also* CALL 15
- BASIC floating point to SLC 16-bit signed integer, CALL 24, 12-15
 - See also* CALL 14
- BASIC module
 - ASCII terminal interface, A-5
 - BASIC Development Software, 2-9
 - BASIC Development Software (PBASE), 2-9
 - compatibility, SOC-1
 - DF1 communication interface, 2-10
 - DF1 protocol configurations, A-9
 - DH-485 communication interface, 2-11
 - diagnostic features, A-4
 - disassembling, 3-3
 - features, A-1
 - floating point, 8-9
 - See also* CALLs 88 and 89
 - hardware features, A-2
 - installing, 1-11
 - memory organization, 5-1
 - network configurations, A-7
 - placement, 1-9
 - programming, 4-1
 - reassembling, 3-11
 - removing from I/O chassis, 3-2
 - software features, A-3
- BASIC module buffers, clear, CALL 120, 13-57
- BASIC module input buffer offsets, 5-1
 - See also* BTR buffer
- BASIC module output buffer offsets, 5-1
 - See also* BTW buffer
- BASIC-52, A-3
- battery, 3-8
 - See also* CALL 80; JW7
 - Allen-Bradley 1770-XY, 3-8
 - installing, 3-8
 - location on board, 3-9
 - MAXELL ER3STC, 3-8
 - replacing, 3-8
 - Tadiran 15-51-03-210-000, 3-8
- battery backup, A-2
- battery condition, 13-9
 - See also* JW7
- battery enable, JW7, 1-8

- See also* CALLs 73, 74, 80
 - battery-backed RAM disable, CALL 73, 13-5
 - See also* JW7
 - battery-backed RAM enable, CALL 74, 13-5
 - See also* JW7
 - baud rates. *See* communication rates
 - BCC error checking, CALL 108, 13-38
 - bitwise operations, truth table, 9-7
 - bitwise operators, 9-7
 - .AND., 9-7
 - .OR., 9-8
 - .XOR., 9-8
 - block-transfer
 - asynchronous, 5-10
 - bi-directional, 5-5
 - CALL 2, 12-2
 - CALL 3, 12-3
 - CALL 4, 12-4
 - CALL 5, 12-4
 - CALL 6, 12-5
 - CALL 7, 12-5
 - calls, 5-5
 - input bits, 1-6
 - See also* data tables
 - output bits, 1-6
 - See also* data tables
 - PLC-2 processor, 5-8
 - PLC-2 sample ladder logic, 5-8
 - PLC-3 processors, 5-9
 - PLC-3 sample ladder logic, 5-9
 - PLC-5 sample ladder logic, 5-10, 5-11
 - PLC-5/250 processors, 5-12
 - PLC-5/250 sample ladder logic, 5-12
 - programming, 5-1
 - programming hints, 5-6
 - sample BASIC program, 5-7
 - support calls, 7-3
 - See also* Chapter 5
 - synchronous, 5-11
 - block-transfer communication, A-3
 - block-transfer-read buffer. *See* BTR buffer
 - block-transfer-write buffer, SOC-3
 - See also* BTW buffer
 - CALL 6, 12-5
 - block-transfers
 - asynchronous, 5-3
 - handshaking, 5-2
 - synchronous, 5-3
 - block-transfer-read buffer, 5-4
 - See also* CALLs 2, 5, 7
 - block-transfer-write buffer, 5-4
 - See also* CALLs 3, 4, 6
 - block-transfer-read buffer, CALL 7, 12-5
 - BRKPNT, 10-2
 - BASIC command, 6-4
 - See also* CONT
 - BTR buffer, 5-1, 5-4
 - See also* CALLs 2, 5, 7; input buffer
 - definition, Using-3
 - transfer data from PRT1 or PRT2 port to, 12-23
 - BTW buffer, 5-1, 5-4
 - See also* CALLs 3, 4, 6; output buffer
 - definition, Using-3
 - transfer data to PRT1 or PRT2 from, 12-29
- ## C
- C programming, 4-3
 - C tool kit, SOC-3
 - ordering, A-13
 - C-stack, 8-1, C-3
 - See also* control stack
 - cable connector
 - DH485 port, 2-3
 - PRT1 and PRT2, 2-2
 - cable lengths, 2-6

- cable pinout
 - DB25, 2-2
 - RJ45, 2-3
- cables, 2-11
- calendar, A-2
- calendar calls, 7-4
- CALL 0, reset module, 12-2
- CALL 1, no operation, 12-2
- CALL 2,
 - timed-block-transfer-read buffer, 12-2
- CALL 3,
 - timed-block-transfer-write buffer, 12-3
- CALL 4, set block-transfer-write length, 12-4
- CALL 5, set block-transfer-read length, 12-4
- CALL 6, block-transfer-write buffer, 12-5
- CALL 7, block-transfer-read buffer, 12-5
- CALL 8, no operation, 12-6
- CALL 9, no operation, 12-6
- CALL 10, 3-digit signed BCD to floating point, 12-6
 - See also* CALL 20
- CALL 11, 16-bit binary to BASIC floating point, 12-7
 - See also* CALL 21
- CALL 12, 4-digit signed octal to BASIC floating point, 12-7
 - See also* CALL 22
- CALL 13, 6-digit signed, fixed decimal BCD to BASIC floating point, 12-8
 - See also* CALL 23
- CALL 14, SLC 16-bit signed integer, 12-8
 - See also* CALL 24
- CALL 15, SLC 16-bit unsigned integer to BASIC floating point, 12-9
 - See also* CALL 25
- CALL 16, enable/disable DF1 packet interrupt, 12-10
- CALL 17, 4-digit BCD to BASIC floating point, 12-11
 - See also* CALL 27
- CALL 18, re-enable control C break function, 12-11
 - See also* CALL 19
- CALL 19, disable control C break function, 12-12
 - See also* CALL 18
- CALL 20, BASIC floating point to 3-digit, signed, fixed decimal BCD, 12-12
 - See also* CALL 10
- CALL 21, BASIC floating point to 16-bit binary, 12-13
 - See also* CALL 11
- CALL 22, BASIC floating point to 4-digit signed octal, 12-13
 - See also* CALL 12
- CALL 23, BASIC floating point to 6-digit signed fixed decimal BCD, 12-14
 - See also* CALL 13
- CALL 24, BASIC floating point to SLC 16-bit signed integer, 12-15
 - See also* CALL 14
- CALL 25, BASIC floating point to SLC 16-bit binary, 12-16
 - See also* CALL 15
- CALL 26, BASIC floating point to 3.3-digit signed BCD, 12-17
 - See also* CALL 39
- CALL 27, BASIC floating point to 4-digit BCD, 12-17
 - See also* CALL 17
- CALL 28, undefined, 12-18
- CALL 29, read/write to a PLC/SLC from the BASIC module internal string, 12-18
- CALL 30, PRT2 port support parameter set, 12-20
 - See also* MODE

- CALL 31, display PRT2 port parameters, 12-21
- CALL 32, enable/disable processor interrupt, 12-22
- CALL 33, transfer data from PRT1 or PRT2 to the BTR buffer, 12-23
- CALL 34, transfer data from BTW buffer to PRT1 or PRT2, 12-29
- CALL 35, retrieve numeric input character from ASCII port, 12-34
See also GET@
- CALL 36, retrieve number of characters in the PRT2 port buffers, 12-35
- CALL 37, clear PRT2 port buffers, 12-35
- CALL 38, expanded ONERR restart, 12-36
See also ONERR
- CALL 39, 3.3-digit signed, fixed decimal BCD to BASIC floating point, 12-38
See also CALL 26
- CALL 40, set the wall clock time, 12-39
- CALL 41, set the wall clock date, 12-40
- CALL 42, set the wall clock day of week, 12-40
- CALL 43, date/time retrieve string, 12-41
- CALL 44, date retrieve numeric, 12-41
- CALL 45, time retrieve string, 12-42
- CALL 46, time retrieve numeric, 12-42
- CALL 47, day of week retrieve string, 12-43
- CALL 48, day of week retrieve numeric, 12-43
- CALL 49, read remote DH-485 SLC data file, 12-44
- CALL 50, write to remote DH-485 SLC data file, 12-50
- CALL 51, undefined, 12-58
- CALL 52, date retrieve string, 12-58
- CALL 53, undefined, 12-58
- CALL 54, undefined, 12-58
- CALL 55, undefined, 12-58
- CALL 56, undefined, 12-58
- CALL 57, undefined, 12-58
- CALL 58, undefined, 12-58
- CALL 59, undefined, 12-58
- CALL 60, string repeat, 12-59
- CALL 61, string append, 12-60
- CALL 62, number to string conversion, 12-61
- CALL 63, string to number conversion, 12-62
- CALL 64, find a string in a string, 12-63
- CALL 65, replace a string in a string, 12-64
- CALL 66, insert a string in a string, 12-65
- CALL 67, delete string from a string, 12-66
- CALL 68, determine length of string, 12-67
- CALL 69, undefined, 13-2
- CALL 70, ROM to RAM program transfer, 13-2
- CALL 71, ROM/RAM to ROM program transfer, 13-3
- CALL 72, ROM/RAM return, 13-4
- CALL 73, battery-backed RAM disable, 13-5
See also JW7
- CALL 74, battery-backed RAM enable, 13-5
See also JW7
- CALL 75, undefined, 13-5
- CALL 76, undefined, 13-5

- CALL 77, protected variable storage, 13-6
See also LD@, MTOP, ST@
- CALL 78, set program port communication rate, 13-8
- CALL 79, no operation, 13-8
- CALL 80, check battery condition, 13-9
See also JW7
- CALL 81, user PROM check and description, 13-10
- CALL 82, check user memory module map, 13-11
- CALL 83, display DH485 port parameters, 13-11
- CALL 84, transfer DH-485 common interface file to BASIC input buffer, 13-12
- CALL 85, transfer BASIC output buffer to DH-485 common interface file, 13-13
- CALL 86, check DH-485 interface file remote write status, 13-14
- CALL 87, check DH-485 interface file remote read status, 13-15
- CALL 88, BASIC floating point to PLC-5 floating point, 13-16
See also CALL 89
- CALL 89, PLC-5 floating point to BASIC floating point, 13-17
See also CALL 88
- CALL 90, read remote DH-485 data file to BASIC input buffer, 13-18
- CALL 91, write BASIC output buffer to remote DH-485 data file, 13-22
- CALL 92, read remote DH-485 common interface file to BASIC input buffer, 13-26
- CALL 93, write output buffer to remote DH-485 common interface file, 13-29
- CALL 94, display current PRT1 port setup, 13-32
- CALL 95, number of characters in PRT1 buffers, 13-32
- CALL 96, clear PRT1 buffers, 13-33
- CALL 97, enable port PRT2 DTR signal, 13-33
See also CALL 98
- CALL 98, disable port PRT2 DTR signal, 13-34
See also CALL 97
- CALL 99, reset print head pointer, 13-34
- CALL 100, download and program assembly language code to EEPROM, 13-35
- CALL 101, upload user EEPROM code to host, 13-35
- CALL 102, undefined, 13-35
- CALL 103, print PRT1 transmit buffer and pointer, 13-36
- CALL 104, print PRT1 receive buffer and pointer, 13-37
- CALL 105, reset PRT1 to default settings, 13-37
- CALL 106, undefined, 13-38
- CALL 107, undefined, 13-38
- CALL 108, enable DF1 driver communications, 13-38
See also JW4, CALL 113
- CALL 109, print argument stack, 13-44
- CALL 110, print ASCII port output buffer and pointer, 13-45
- CALL 111, print PRT2 port receive buffer and pointer, 13-46
- CALL 112, user LED control, 13-47
- CALL 113, disable DF1 driver communications, 13-47
See also JW4, CALL 108
- CALL 114, transmit DF1 packet, 13-48
- CALL 115, check DF1 status, 13-49

- CALL 116, call user defined assembly language routine, 13-50
- CALL 117, DF1 packet length, 13-51
- CALL 118, PLC/SLC unsolicited writes, 13-52
- CALL 119, reset PRT2 port to default settings, 13-56
- CALL 120, clear BASIC module I/O buffers, 13-57
- CALL 121, undefined, 13-57
- CALL 122, read remote DF1 PLC data file, 13-58
- CALL 123, write remote DF1 PLC data file, 13-66
- CALL 124, undefined, 13-74
- CALL 125, undefined, 13-74
- CALL 126, undefined, 13-74
- CALL 127, undefined, 13-74
- call user defined assembly language routine, CALL 116, 13-50
- calls, 4-2, 12-1, 13-1
 - See also* Chapters 12 and 13
 - background operations, 7-6
 - backplane conversion, 7-4
 - See also* Chapter 8
 - block-transfer, 7-3
 - See also* Chapter 5
 - calendar, 7-4
 - clock, 7-4
 - command line, 7-7, 10-22
 - DF1 protocol, 7-6
 - DH-485 communication, 7-5
 - execution control, 7-7
 - input, 7-8
 - interrupt support, 7-7
 - math, 7-4
 - See also* Chapter 8
 - memory manipulation, 7-1
 - miscellaneous, 7-2
 - new, SOC-4
 - no operation, SOC-5
 - output, 7-9
 - port communication, 7-2
 - quick reference, E-1
 - redefined, SOC-5
 - setup, 7-9
 - status, 7-10
 - string, 7-5
 - See also* Chapter 8
 - unsupported, SOC-5
 - using, 7-1
 - See also* Chapters 12 and 13
- CAN'T CONTINUE, C-3
- cassette recorder, SOC-5
- CBY, special function operator, 9-18
- chassis addressing. *See* addressing
- check battery condition, CALL 80, 13-9
 - See also* JW7
- check DF1 status, CALL 115, 13-49
- check DH-485 interface file remote write status, CALL 86, 13-14
- check DH-485 interface file remote read status, CALL 87, 13-15
- check user memory module map, CALL 82, 13-11
- chip insertion tool, 3-7
- CHR, string operator, 9-16
 - See also* STRING, CALL 65
- CIF, definition, Using-3
- CLEAR, 11-2
- clear BASIC module I/O buffers, CALL 120, 13-57
- clear PRT1 buffers, CALL 96, 13-33
- clear PRT2 port buffers, CALL 37, 12-35
- CLEARI, 11-3
 - See also* ONTIME
- CLEARs, 11-3
- clock, A-2

- set date, CALL 41, 12-40
- set day of week, CALL 42, 12-40
- set time, CALL 40, 12-39
- clock calls, 7-4
- clock/calendar accuracy, A-12
- CLOCK0, 11-4
 - See also* ONTIME, TIME
- CLOCK1, 11-5
 - See also* ONTIME, TIME
- command
 - BRKPNT, 6-4, 10-2
 - See also* CONT
 - CONT, 10-3
 - Control C, 10-4
 - See also* CALLs 18 and 19
 - Control Q, 10-5
 - See also* Control S
 - Control S, 10-6
 - See also* Control Q
 - EDIT, 10-7
 - ERASE, 10-8
 - See also* PROG
 - LIST, 10-9
 - LIST#, 10-9
 - LIST@, 10-9
 - MODE, 11-20
 - NEW, 10-10
 - NULL, 10-10
 - PROG, 10-11
 - PROG1, 10-12
 - PROG2, 10-13
 - RAM, 10-15
 - REN, 10-16
 - ROM, 10-17
 - RROM, 10-18
 - RUN, 10-19
 - SNGLSTP, 6-4, 10-20
 - See also* CONT
 - VER, 10-21
 - XFER, 10-21
- command line calls, 7-7, 10-22
- command mode, 4-1
 - commands, 4-1, 10-1
 - See also* Chapter 10
 - quick reference, E-1
 - common interface file
 - check read status, CALL 87, 13-15
 - check write status, CALL 86, 13-14
 - read to input buffer, CALL 92, 13-26
 - transfer output buffer to DH-485 CIF, CALL 85, 13-13
 - transfer to BASIC module input buffer, CALL 84, 13-12
 - write output buffer to remote DH-485, CALL 93, 13-29
 - communicating with SLC processors, 2-3
 - communication cable, DH-485 network, 2-11
 - communication cables, ordering, A-13
 - communication mode
 - RS-232, 2-2
 - RS-422, 2-2
 - RS-485, 2-2
 - communication modes, 2-2, 2-8
 - communication networks, A-3
 - communication ports, 2-1
 - overview, 2-1
 - communication rate, program port, CALL 78, 13-8
 - communication rates, 2-6
 - cable lengths, 2-6
 - DH485 port, 2-6
 - PRT1 port, 2-6
 - PRT2 port, 2-6
 - compatibility, SOC-1
 - with 1746-BAS, SOC-5
 - with Series A, SOC-1
 - complex expression, 9-1
 - components needed for DF1 communication
 - dial-up modem, 2-10

- leased phone line, 2-10
 - radio link, 2-10
 - components needed for DH-485
 - communication
 - 1747-AIC Isolated Link Coupler, 2-11
 - 1747-KE DH-485/RS-232C Communication Interface Module, 2-11
 - 1747-PIC Interface/Converter, 2-11
 - 1770-KF3 DH-485 Communication Interface Module, 2-11
 - 1784-KR DH-485 Interface Card, 2-11
 - concatenation, CALL 61, 12-60
 - configuration jumpers, 1-3
 - JW1, watchdog timer, 1-4
 - JW2, memory module, 1-4
 - JW3, CPU speed, 1-5
 - JW4, operating mode, 1-5
 - JW5, backplane configuration, 1-6
 - JW6, PRT2 communication rate, 1-7
 - See also* PROG1, PROG2, MODE
 - JW7, battery enable, 1-8
 - See also* CALLs 73, 74, 80
 - JW8, PRT1 configuration, 1-8
 - See also* communication modes
 - JW9, PRT2 configuration, 1-8
 - See also* communication modes
 - configuration plugs, Series A, D-1
 - connect peripheral devices, 1-12
 - console device, definition, Using-3
 - constants, 9-1
 - CONT, 10-3
 - contacting Allen-Bradley for assistance, Using-6
 - Control C, 10-4
 - See also* CALLs 18 and 19
 - Control Q, 10-5
 - See also* Control S
 - Control S, BASIC command, 10-6
 - See also* Control Q
 - control stack, 8-1, 11-8, 11-9, 11-11, 11-13
 - See also* CLEARS, C-Stack
 - conventions in this manual, Using-4
 - conversions, numeric, 8-4
 - COS, trigonometric function, 9-10
 - CPU speed, SOC-3
 - JW3, 1-5
 - normal, 1-5
 - turbo, 1-5
 - CRC error checking, CALL 108, 13-38
 - creating a program, 4-2
- D**
- DATA, 11-6
 - See also* READ, RESTORE
 - Data Communication Equipment (DCE). *See* DCE
 - data recorder, SOC-5
 - data storage, A-3
 - data tables, 5-2
 - input image table, 5-3
 - output image table, 5-2
 - Data Terminal Equipment (DTE). *See* DTE
 - data type generation, A-3
 - data types, 8-1
 - send to BASIC module, 8-4
 - send to PLC processor, 8-4
 - date
 - retrieve numeric, CALL 44, 12-41
 - retrieve string
 - CALL 43, 12-41

- CALL 52, 12-58
 - setting, CALL 41, 12-40
- date retrieve numeric, CALL 44, 12-41
- date/time retrieve string, CALL 43, 12-41
- day of week
 - retrieve numeric, CALL 48, 12-43
 - retrieve string, CALL 47, 12-43
 - set, CALL 42, 12-40
- day of week retrieve numeric, CALL 48, 12-43
- day of week retrieve string, CALL 47, 12-43
- DB25 female cable connector, 2-2
- DBY, special function operator, 9-18
- DCE, cable pinout, 2-5
- DCE and DTE, overview, 2-5
- debugging a program, 6-4
 - See also* BRKPNT, SNGLSTP, STOP
- decimal conversion table, B-1
- definitions, Using-3
- delete string from a string, CALL 67, 12-66
- deleting a program line, 6-3
- determine length of string, CALL 68, 12-67
- DF1, components required for, 2-10
- DF1 data file, read remote, CALL 122, 13-58
- DF1 driver communications, enable, CALL 108, 13-38
- DF1 protocol, SOC-2, 2-10
 - See also* JW4, CALL 108
 - CALL 16, 12-10
 - check status, CALL 115, 13-49
 - components required for, A-9
 - configuration jumper JW4, 1-5
 - configurations, A-9
 - definition, Using-3
 - disable drive communications, CALL 113. *See* JW4, CALL 108
 - enable driver communications, CALL 108, 13-38
 - ONDF1, 11-22
 - packet length, CALL 117, 13-51
 - transmit packet, CALL 114, 13-48
 - write remote data file, CALL 123, 13-66
- Df1 protocol calls, 7-6
- DF1 protocol port, PRT2 port, 2-10, A-9
- DH-485
 - common interface file, CALL 86, 13-14
 - definition, Using-3
 - network, SOC-2
- DH-485 communication calls, 7-5
- DH-485 interface, A-6
- Dh-485 interface card, ordering, A-13
- DH-485 network
 - 1747-AIC Isolated Link Coupler, 2-12, A-7
 - 1747-PIC Interface/Converter, 2-12, A-7
 - cable requirements, 2-11
 - check interface file remote read status, 13-15
 - check interface file remote write status, CALL 86, 13-14
 - common interface file, CALL 87, 13-15
 - configurations, A-7
 - interface card 1784-KR, 2-13
 - interfacing with BASIC module and development software, 2-11
 - programming interface, 4-5
 - read remote common interface file to BASIC input buffer, CALL 92, 13-26

- read remote data file, CALL 49, 12-44
- read remote data file to BASIC input buffer, CALL 90, 13-18
- serial communication link
 - CALL 86, 13-14
 - CALL 87, 13-15
- transfer data to BASIC input buffer, 13-12
- write BASIC output buffer to remote data file, CALL 91, 13-22
- write output buffer to remote common interface file, CALL 93, 13-29
- write to remote data file, 12-50
- Dh-485 network, transfer BASIC output buffer to common interface file, 13-13
- DH-485 network, communication cables, 2-11
- DH-485, network, 12-53
- DH485 port, SOC-2, A-2
 - cable pinout, 2-3
 - communication calls, 7-5
 - communication mode, 2-3
 - communication rates, 2-6
 - connecting peripherals, 1-12
 - display parameters, CALL 83, 13-11
 - electrical isolation, 2-3
 - network port, 2-11
 - operating mode, JW4, 1-5
 - port isolation, A-12
 - program port, 2-9, 4-5
 - communication rate, CALL 78, 13-8
 - programming interface, A-6
 - set parameters, 11-20
 - See also* CALL 83
- diagnostic features, A-4
- dial-up modem for DF1 communication, 2-10
- DIM, 11-7
- dimensioned variable, definition, Using-3
- dimensioned variables, 9-2
- disable battery-backed RAM, CALL 73, 13-5
 - See also* JW7, CALL 74
- disable control C break function, CALL 19, 12-12
 - See also* CALL 18
- disable DF1 driver communications, CALL 113, 13-47
 - See also* CALL 108
- disable port PRT2 DTR signal, CALL 98, 13-34
- disable/enable DF1 packet interrupt, CALL 16, 12-10
- disable/enable processor interrupt, CALL 32, 12-22
- disassemble module, 3-3
- discrete transfers, 5-2
- display current PRT1 port setup, CALL 94, 13-32
- display DH485 port parameters, CALL 83, 13-11
- display PRT2 port parameters, CALL 31, 12-21
- disposing lithium battery, 3-8
- divide (/), arithmetic operator, 9-5
- DIVIDE BY ZERO, C-3
- DO-UNTIL, 11-8
- DO-WHILE, 11-9
- documentation, Using-4
 - Application Considerations for Solid-State Controls (SGI-1.1), Using-4
 - Automation Glossary (ICCG-7.1), Using-4
 - BASIC Development Software (1746-6.2), Using-4
 - comments on, Using-6

- DH-485/RS-232G Interface Module (1747-NU001), Using-4
- DH, DH+, DH-485 Protocol and Command Set Reference Manual, Using-4
- National Electrical Code, Using-4
- problems with, Using-6
- Programmable Controller Grounding and Wiring Guidelines (1770-4.1), Using-4
- Publication Index (SD499), Using-4
- download assemble language code to EEPROM, CALL 100, 13-35
- DPD selection, CALL 108, 13-38
- DTE, cable pinout, 2-5
- DTE and DCE, overview, 2-5
- DTR signal
 - disable, 13-34
 - enable, 13-33
- E**
- EDIT, 10-7
- editing a program line, 6-1
 - See also* EDIT
- editing operations, 6-2, 10-7
 - See also* EDIT
- EEPROM, SOC-1
 - See also* memory module definition, Using-3
- electric isolation
 - DH485, 2-3
 - PRT1, 2-2
 - PRT2, 2-2
- electrical interface. *See* JW8 and JW9
- electrostatic discharge, 1-1
- enable battery-backed RAM, CALL 74, 13-5
 - See also* JW7, CALL 73
- enable DF1 driver communications, CALL 108, 13-38
 - See also* JW4, CALL 113
- enable port PRT2 DTR signal, CALL 97, 13-33
- enable/disable DF1 packet interrupt, CALL 16, 12-10
- enable/disable processor interrupt, CALL 32, 12-22
- END, 11-10
- enhanced serial port modifiers, SOC-3, 9-17
- entering a program, 4-7
 - See also* LIST
- environmental conditions, A-11
- EOF, special function operator, 9-17
- EPROM
 - See also* memory module definition, Using-3
- equal to (=), relational operator, 9-9
- ERASE, 10-8
 - See also* PROG
- error checking
 - BCC, CALL 108, 13-38
 - CRC, CALL 108, 13-38
- error messages
 - from BASIC module, C-2
 - from CALL routines, C-4
 - memory support, C-5
 - miscellaneous, C-6
 - PRT2 support, C-4
 - string support, C-5
 - wall clock, C-4
- error trapping
 - CALL 38, 12-36
 - ONERR, 11-23
- errors, arithmetic, 9-6
 - See also* troubleshooting, ONERR, CALL 38
- error-trapping support, SOC-2
- ESD. *See* electrostatic discharge

- example block transfer BASIC program, 5-7
 - example ladder logic
 - CALL 118, 13-56
 - CALL 122, 13-65
 - CALL 123, 13-74
 - CALL 32, 12-22
 - CALL 33, 12-28
 - CALL 34, 12-33
 - CALL 49, 12-49
 - CALL 50, 12-57
 - PLC-2 processor, 5-8
 - PLC-3 processor, 5-9
 - PLC-5 processor, 5-10, 5-11
 - PLC-5/250 processor, 5-12
 - exclusive OR, 9-8
 - execution control calls, 7-7
 - EXP, logarithmic operator, 9-13
 - See also* **
 - expanded ONERR restart, CALL 38, 12-36
 - See also* ONERR
 - exponentiation (**), arithmetic operator, 9-5
 - See also* EXP
 - expressions, 9-1
 - EXTRA IGNORED, C-3, C-5
- F**
- find a string in a string, CALL 64, 12-63
 - floating-point
 - BASIC module, 8-9
 - See also* CALLs 88 and 89
 - PLC-5, 8-9
 - See also* CALLs 88 and 89
 - floating-point numbers, 8-3
 - FOR-TO-NEXT, 11-11
 - FREE, special function operator, 9-17
 - free-running clock, A-2
 - full-duplex slave mode, CALL 108, 13-38
 - functional operators, 9-11
- ABS, 9-11
 - INT, 9-12
 - NOT, 9-11
 - PI, 9-12
 - RND, 9-12
 - SGN, 9-12
 - SQR, 9-12
- G**
- GET, 11-12
 - get DF1 packet length, CALL 117, 13-51
 - get number of characters in PRT2 port buffers, CALL 36, 12-35
 - get numeric input character from ASCII port, CALL 35, 12-34
 - See also* GET@
 - GET#, 11-12
 - GET@, 11-12
 - getting started flowchart, Using-5
 - GOSUB, 11-13
 - GOTO, 11-14
 - greater than (>), relational operator, 9-9
 - greater than or equal to (>=), relational operator, 9-9
 - guard against electrostatic discharge, 1-1
- H**
- half-duplex slave operation, CALL 108, 13-38
 - handshaking, 2-4
 - block-transfers, 5-2
 - enabling/disabling, 11-20
 - hardware, 2-5
 - modem full-duplex mode, CALL 108, 13-42
 - modem half-duplex mode, CALL 108, 13-40
 - modem handshaking selection, CALL 108, 13-38
 - software handshaking, 2-4
 - hardware features, A-2

- hardware handshaking, 2-5, 2-8, 11-27
- hardware specification, power requirements, A-11
- hardware specifications
 - clock/calendar accuracy, A-12
 - environmental conditions, A-11
 - maximum communication distances, A-12
 - port isolation, A-12
- hardware handshaking, 11-29
- hex conversion table, B-1
- hierarchy of operations, 9-3
- hints, programming, 4-2
- how to use manual, Using-2

- I**
- I/O chassis, 1-9
 - inserting module, 1-11
 - removing module, 3-2
- IDLE, 11-14
- IF-THEN-ELSE, 11-15
- INCOMPLETE ROM PROGRAM FOUND, C-6
- indicator lights, 1-13, A-4, C-1
- initiate single step execution, 6-4
 - See also* CONT
- INPL, 11-16
 - See also* INPL, INPUT
- INPL#, 11-16
- INPL@, 11-16
- INPS, 11-16
 - See also* INPL, INPUT
- INPS#, 11-16
- INPS@, 11-16
- INPUT, 11-17, C-5
 - See also* INPL, INPS
- input buffer
 - See also* BTR buffer, receive buffer
 - clear, CALL 120, 13-57
 - definition, Using-3
 - offsets, 5-1
 - See also* BTR buffer
 - read remote common interface file to BASIC input buffer, CALL 92, 13-26
 - read remote data file to BASIC input buffer, CALL 90, 13-18
 - transfer DH-485 CIF to, CALL 84, 13-12
- input calls, 7-8
- input image table, 5-3
- INPUT#, 11-17
- INPUT@, 11-17
- insert a string in a string, CALL 66, 12-65
- insert module into I/O chassis, 1-11
- installing
 - BASIC module, 1-1, 1-11
 - battery, 3-8
 - memory modules, 1-2, 3-5
- installing components, 3-1
- INSUFFICIENT NUMBER OF STRING CHARACTERS, C-4
- INSUFFICIENT NUMBER OF STRING CHARACTERS ALLOCATED, C-4
- INSUFFICIENT STRING SIZE, C-5
- INT, functional operator, 9-12
- integer numbers, 8-3
- interface/converter, ordering, A-13
- internal string, 8-2
- interpreter mode, 4-1
- interrupt routines, exiting with RETI, 11-33
- interrupt support calls, 7-7
- interrupts
 - enable/disable DF1-CALL 16, 12-10
 - enable/disable processor, CALL 32, 12-22
- INVALID BAUD RATE ENTERED, C-6

- INVALID DATE/TIME PUSHED, C-4
- INVALID INPUT DATA, C-4
- INVALID MTOP ADDRESS ENTERED, C-5
- INVALID NUMBER PUSHED, C-4
- INVALID VALUE PUSHED, C-4
- J**
- JEDEC standard, memory modules, 3-4
- JW1, watchdog timer, 1-4
- JW2, memory module, 1-4
- JW3, CPU speed, 1-5
- JW4, operating mode, 1-5
- JW5, backplane configuration, 1-6
- JW6, PRT2 communication rate, 1-7
See also PROG1, PROG2, MODE
- JW7, battery enable, 1-8
See also CALLs 73, 74, 80
- JW8, PRT1 configuration, 1-8
- JW9, PRT2 configuration, 1-8
See also communication modes
- K**
- keying, 1-10, A-11
- L**
- ladder logic
- CALL 118, 13-56
 - CALL 122, 13-65
 - CALL 123, 13-74
 - CALL 32, 12-22
 - CALL 33, 12-28
 - CALL 34, 12-33
 - CALL 49, 12-49
 - CALL 50, 12-57
- LD@, 11-18
See also ST@, CALL 77
- leased phone lines for DF1 communication, 2-10
- LED indicators, SOC-1, A-2, A-4, C-1
- reading, 1-13
See also troubleshooting user control, CALL 112, 13-47
- LEN, 10-17, 10-18
- special function operator, 9-17
- length of string, CALL 68, 12-67
- less than (<), relational operator, 9-9
- less than or equal to (<=), relational operator, 9-9
- LET, 11-19
- lights, A-4, C-1
See also LED indicators
- link coupler, ordering, A-13
- LIST, 10-9
- LIST #, 10-9
- LIST @, 10-9
- lithium battery condition, 13-9
See also JW7
- lithium battery disposal, 3-8
- LOG, logarithmic operator, 9-13
- logarithmic operators, 9-13
- EXP, 9-13
See also **
 - LOG, 9-13
- logical bitwise operators, 9-7
- logical operators
- .AND., 9-7
 - .OR., 9-8
 - .XOR., 9-8
- M**
- manual organization, Using-2
- math calls, 7-4
See also Chapter 8
- math precision, A-12
- maximum communication distances, A-12
- memory modules
- PROG1, 10-12
 - PROG2, 10-13
- MEMORY ALLOCATION, C-3

- memory manipulation calls, 7-1
- memory module
 - configuration JW2, 3-7
 - JW2, 1-4
 - locations on board, 3-6
- memory modules, SOC-1, 3-4, A-2, A-10
 - See also* RAM, ROM, XFER, PROG, PROG1, PROG2
 - check and description, CALL 81, 13-10
 - chip speed, 3-7
 - See also* JW3
 - configuration jumper, 1-4
 - definition, Using-3
 - download/program assembly language code, 13-35
 - ERASE, 10-8
 - See also* PROG
 - installing, 1-2, 3-5
 - JEDEC standards, 3-4
 - memory map, CALL 82, 13-11
 - ordering, A-13
 - PROG, 10-11
 - replacing, 3-5
 - ROM, 10-17
 - SKT1, 3-6
 - SKT2, 3-6
 - upload user code to host, 13-35
 - with carriers, 3-5, 3-6
 - without carriers, 3-5, 3-6
- memory organization, 5-1
- memory requirements, A-10
- miscellaneous calls, 7-2
- MODE, 11-20
- mode
 - command, 4-1
 - interpreter, 4-1
 - program, 4-1
 - run, 4-1
- module keying, A-11
- module location, 1-9, A-11
- MTOP, 10-17, 10-18
- CALL 77, 13-6
- definition, Using-3
- PROG1, 10-12
- special function operator, 9-18
 - See also* CALL 77
- multidrop, 2-2
- multiply (*), arithmetic operator, 9-5
- N**
- negation (-), arithmetic operator, 9-6
- network configurations, A-7
- network port
 - See also* DH485 port
 - configuration jumper JW4, 1-5
 - definition, Using-3
 - DH485 port, 2-11
- NEW, 10-10
- new calls, SOC-4
- NEXT, 11-21
 - See also* FOR-T0-NEXT
- NO DATA, C-3
- NO PROGRAM FOUND BUT THE PROM IS NOT BLANK, C-6
- non-ASCII data. *See* software handshaking
- normal mode, 1-5
- NOT, functional operator, 9-11
- not equal to (\neq), relational operator, 9-9
- NULL, 10-10
- NUMBER BYTES/STRING EXCEED 254, C-4
- number of characters in PRT1 buffers, CALL 95, 13-32
- number to string conversion, CALL 62, 12-61
- numbering program lines, 4-6
- numeric conversion
 - CALL 10, 12-6
 - CALL 11, 12-7
 - CALL 12, 12-7

- CALL 13, 12-8
 - CALL 14, 12-8
 - CALL 15, 12-9
 - CALL 17, 12-11
 - CALL 20, 12-12
 - CALL 21, 12-13
 - CALL 22, 12-13
 - CALL 23, 12-14
 - CALL 24, 12-15
 - CALL 25, 12-16
 - CALL 26, 12-17
 - CALL 27, 12-17
 - CALL 39, 12-38
 - CALL 88, 13-16
 - CALL 89, 13-17
 - numeric conversions, 8-4
 - numeric data types, 8-3
- O**
- octal conversion table, B-1
 - ON-GOSUB, 11-24
 - ON-GOTO, 11-26
 - ONDF1, 11-22
 - See also* CALL 16
 - ONERR, 11-23
 - See also* CALL 38
 - ONTIME, 11-25
 - See also* CLOCK1, TIME
 - operating mode, JW4, 1-5
 - operating modes, 2-7
 - operational codes
 - full-duplex mode, CALL 108, 13-41
 - half-duplex mode, CALL 108, 13-39
 - operators, 4-1, 9-3
 - See also* Chapter 9
 - arithmetic, 9-5
 - bitwise, 9-7
 - functional, 9-11
 - logarithmic, 9-13
 - logical bitwise, 9-7
 - quick reference, E-1
 - relational, 9-9
 - special function, 9-17
 - string, 9-14
 - trigonometric, 9-10
 - operating modes
 - ASCII port, 2-7
 - DF1 protocol port, 2-7
 - network port, 2-7
 - program port, 2-7
 - OR, 9-8
 - order of operations, 9-3
 - output buffer
 - See also* BTW buffer, transmit buffer
 - clear, CALL 120, 13-57
 - definition, Using-3
 - offsets, 5-1
 - See also* BTW buffer
 - transfer DH-485 CIF to, CALL 85, 13-13
 - write BASIC output buffer to remote data file, CALL 91, 13-22
 - output calls, 7-9
 - output image table, 5-2
 - overflow and division by zero, arithmetic operator, 9-6
 - See also* ONERR, CALL 38
- P**
- PBASE, SOC-3, Using-3
 - See also* BASIC Development Software
 - RS-232 interface, 2-9
 - PH1., 11-27
 - PH1.#, 11-27
 - PH1.~, 11-27
 - PHO., 11-27
 - PHO.#, 11-27
 - PHO.~, 11-27
 - PI, functional operator, 9-12
 - pinout
 - DB25, 2-2
 - RJ45, 2-3

- pinout for connectors, 2-2
- PLC processor
 - definition, Using-3
 - read remote DF1 PLC data file, CALL 122, 13-58
 - unsolicited write, CALL 118, 13-52
 - write to remote DF1 PLC data file, CALL 123, 13-66
- PLC-5 floating point, SOC-3
- PLC-5 floating point to BASIC floating point, CALL 89, 13-17
 - See also* CALL 88
- PLC-5, floating point, 8-9
 - See also* CALLs 88 and 89
- point to point, 2-2
- pointer
 - print PRT1 receive pointer, CALL 104, 13-37
 - print PRT1 transmit pointer, CALL 103, 13-36
 - print PRT2 receive, CALL 111, 13-46
 - print PRT2 transmit, CALL 110, 13-45
 - reset print head pointer, CALL 99, 13-34
- POP, 8-1, 11-28
 - See also* PUSH
- port communication calls, 7-2
- port driver and receiver, A-11
- port isolation, A-12
- power on, flowchart, 10-14
- power requirements, 1-2, A-11
- power up, 1-12
 - See also* PROG2
 - configuration jumper JW4, 1-5
- PRINT, 11-29
- print argument stack, CALL 109, 13-44
- print head, reset pointer, CALL 99, 13-34
- print PRT1 receive buffer and pointer, CALL 104, 13-37
- print PRT1 transmit buffer and pointer, CALL 103, 13-36
- print PRT2 receive buffer and pointer, CALL 111, 13-46
- print PRT2 transmit buffer and pointer, CALL 110, 13-45
- PRINT#, 11-29
- PRINT@, 11-29
- product features, A-1
- product overview, A-1
- PROG, 10-11
 - See also* PROG1, PROG2, MODE, CALLs 81 and 82
- PROG 2, 10-13
 - See also* JW4, PROG, PROG1, MODE
 - BASIC command, 10-13
- PROG1, 10-12
 - See also* PROG, PROG2, MODE
- program assembly language code to EEPROM, CALL 100, 13-35
- program lines
 - maximum character number, 4-6
 - multiple statements, 4-6
- program mode, 4-1
- PROGRAM NOT FOUND, C-5
- program port
 - See also* DH485 port, PRT1
- port
 - calls, 7-2
 - communication rate, 13-8
 - configuration jumper JW4, 1-5
 - definition, Using-3
 - DH485 port, 2-9, 4-5
 - See also* Chapter 2
 - GET, 11-12
 - PRT1 port, 2-9, 4-3
 - See also* Chapter 2
 - RS-232 interface, 4-4
 - See also* Chapter 2
- program storage, A-3
- program transfer

- ROM to RAM, 13-2
- ROM/RAM to ROM, 13-3
- PROGRAMMING, C-3
- programming
 - ASCII terminal emulator, 4-3
 - BASIC Development Software, 4-4
 - BASIC module output buffer offsets, 5-1
 - See also* BTW buffer
 - block-transfers, 5-1
 - calls, 4-2
 - See also* Chapters 12 and 13
 - commands, 4-1
 - See also* Chapter 10
 - creating, 4-2
 - debugging, 6-4
 - See also* BRKPNT, SNGLSTP, STOP
 - deleting a program line, 6-3
 - editing a program line, 6-1
 - See also* EDIT
 - entering program, 4-7
 - See also* LIST
 - hints, 4-2, 5-6, 8-2, 11-32, 11-38, 12-23, 12-29, 12-36, 12-44, 12-50
 - input buffer offsets, 5-1
 - instructions, 4-1
 - numbering lines, 4-6
 - operators, 4-1
 - See also* Chapter 9
 - overview, 4-1
 - renumbering a program, 6-3
 - See also* REN
 - running a program, 4-9
 - See also* RUN
 - setting breakpoints, 6-4
 - See also* CONT
 - single step execution, 6-4
 - See also* CONT
 - statements, 4-2
 - See also* Chapter 11
 - stopping a program, 4-9
 - See also* JW4, CALLs 18 and 19
 - tips, 5-6, 11-38, 12-23, 12-29, 12-36, 12-44, 12-50
- programming interface
 - ASCII terminal, A-5
 - BASIC Development Software, A-6
 - DH-485 interface, A-6
 - RS-232 interface, A-6
- programming interfaces, A-5
- programming languages, A-3
- protected variable storage, CALL 77, 13-6
 - See also* LD@, MTOP, ST@
- PRT1, program port, communication rate, CALL 78, 13-8
- PRT1 port, A-2
 - @ operator, 9-17
 - ASCII port, 2-8
 - cable pinout, 2-2
 - calls, 7-2
 - clear buffers, CALL 96, 13-33
 - communication modes, 2-2
 - See also* JW8
 - communication rates, 2-6
 - See also* PROG1, PROG2, MODE
 - connecting peripherals, 1-12
 - electric isolation, 2-2
 - JW8, 1-8
 - See also* communication modes
 - number of characters in buffers, CALL 95, 13-32
 - operating mode, JW4, 1-5
 - port isolation, A-12
 - print receive buffer and pointer, CALL 104, 13-37
 - print transmit buffer and pointer, CALL 103, 13-36
 - program port, 2-9, 4-3

- See also* Chapter 2
- programming interface, A-5, A-6
- receive buffer, 2-3
 - clear, CALL 96, 13-33
 - number of characters in, CALL 95, 13-32
 - print, CALL 104, 13-37
- reset default settings, CALL 105, 13-37
- RS-232 Interface, 4-4
 - See also* Chapter 2
- set parameters, 11-20
 - See also* CALL 94
- setup, CALL 94, 13-32
- transfer data from BTW buffer, 12-29
- transfer data to BTR buffer, CALL 33, 12-23
- transmit buffer, 2-3
 - clear, CALL 96, 13-33
 - number of characters in, CALL 95, 13-32
 - print, CALL 103, 13-36
- PRT2 communication rate, JW6, 1-7
 - See also* PROG1, PROG2, MODE
- PRT2 port, 2-2, A-2
 - See also* JW9
- # operator, 9-17
- ASCII port, 2-8
- cable pinout, 2-2
- calls, 7-3
- clear buffers, 12-35
- communication rates, 2-6
 - See also* JW6, PROG1, PROG2, MODE
- connecting peripherals, 1-12
- DF1 protocol, 2-10, A-9
 - CALL 16, 12-10
 - check status, CALL 115, 13-49
 - disable driver, CALL 113, 13-47
 - enable driver, CALL 108, 13-38
 - get packet length, CALL 117, 13-51
 - read data file, CALL 122, 13-58
 - transmit packet, CALL 114, 13-48
 - write remote DF1 PLC data file, CALL 123, 13-66
- DF1 protocol communication calls, 7-6
- disable DTR signal, CALL 98, 13-34
- display parameters, CALL 31, 12-21
- electric isolation, 2-2
- enable DTR signal, CALL 97, 13-33
- error messages, C-4
- get number of characters in buffers, CALL 36, 12-35
- JW9, 1-8
 - See also* communication modes
- operating mode, JW4, 1-5
- port isolation, A-12
- print receive buffer and pointer, CALL 111, 13-46
- print transmit buffer buffer and pointer, CALL 110, 13-45
- receive buffer, 2-3
 - clear buffer, CALL 37, 12-35
 - number of characters in, CALL 36, 12-35
 - print, CALL 111, 13-46
- reset to default settings, CALL 119, 13-56
- retrieve numeric input character from, CALL 35, 12-34
 - See also* GET@

- set handshaking, CALL 30, 12-20
 - See also* MODE
- set parameters, 11-20
 - See also* CALL 31
 - CALL 30, 12-20
 - See also* MODE
- transfer data from BTW buffer, 12-29
- transfer data to BTR buffer, 12-23
- transmit buffer, 2-3
 - clear buffer, CALL 37, 12-35
 - number of characters in, CALL 36, 12-35
 - print, CALL 110, 13-45
- PRT2 port support parameter set, CALL 30, 12-20
 - See also* MODE
- publication problem report, Using-6
- purpose of manual, Using-1
- PUSH, 8-1, 11-30
 - See also* POP
- Pyramid Solutions Program, SOC-3, A-13

- Q**
- quick reference, E-1

- R**
- radio link for DF1 communication, 2-10
- RAM, 10-15, A-2
 - See also* ROM
 - available user RAM formula, 10-15
 - disable battery-backed, 13-5
 - See also* JW7, CALL 72
 - enable battery-backed, 13-5
 - See also* JW7
 - program transfer to ROM
 - CALL 70, 13-2
 - CALL 71, 13-3
 - return to, CALL 72, 13-4
- RAM memory, SOC-1, A-10
 - definition, Using-3
- re-enable control C break function, CALL 18, 12-11
 - See also* CALL 19
- READ, 11-31
 - See also* DATA, RESTORE
- read block words, 1-6
- read remote DF1 PLC data file, CALL 122, 13-58
- read remote DH-485 common interface file to BASIC input buffer, CALL 92, 13-26
- read remote DH-485 data file to BASIC input buffer, CALL 90, 13-18
- read remote DH-485 SLC data file, CALL 49, 12-44
- read/write to a PLC/SLC from the BASIC module internal string, CALL 29, 12-18
- reassemble the module, 3-11
- receive, print PRT2 buffer, CALL 111, 13-46
- receive buffer
 - See also* input buffer
 - clear PRT1 buffer, CALL 96, 13-33
 - clear PRT2 buffer, CALL 37, 12-35
 - definition, Using-3
 - number of characters in (PRT2), CALL 36, 12-35
 - print PRT1 buffer, CALL 104, 13-37
 - PRT1 and PRT2, 2-3
 - retrieve number of characters (PRT1), CALL 95, 13-32
 - software handshaking, 2-4
- redefined calls, SOC-5
- related products, A-13
- related publications, Using-4

- relational expressions, 9-1
 - relational operator, 9-9
 - equal to (=), 9-9
 - greater than (>), 9-9
 - greater than or equal to (>=), 9-9
 - less than (<), 9-9
 - less than or equal to (<=), 9-9
 - not equal to (!=), 9-9
 - relational operators, 9-9
 - REM, 11-32
 - remove module from I/O chassis, 3-2
 - REN, 10-16
 - renumbering a program, 6-3
 - See also* REN
 - replace a string in a string, CALL 65, 12-64
 - replacing
 - battery, 3-8
 - memory modules, 3-5
 - replacing components, 3-1
 - reset module, CALL 0, 12-2
 - reset print head pointer, CALL 99, 13-34
 - reset PRT1 to default settings, CALL 105, 13-37
 - reset PRT2 port to default settings, CALL 119, 13-56
 - reset switch, SOC-3, 1-12, 12-2, A-2
 - See also* CALL 0
 - RESTORE, 11-32
 - See also* DATA, READ
 - RETI, 11-33
 - retrieve number of characters in PRT2 port buffers, CALL 36, 12-35
 - retrieve numeric input character from ASCII port, CALL 35, 12-34
 - See also* GET@
 - RETURN, 11-34
 - See also* GOSUB
 - return to ROM/RAM, CALL 72, 13-4
 - RJ45 cable connector, 1747-C10,-C11, 2-3
 - RND, functional operator, 9-12
 - ROM, 10-17
 - See also* RAM, RROM, XFER, CALLs 70, 71, 72
 - ROM memory, definition, Using-3
 - ROM to RAM program transfer, CALL 70, 13-2
 - ROM/RAM return, CALL 72, 13-4
 - ROM/RAM to ROM program transfer, CALL 71, 13-3
 - RROM, 10-18
 - See also* RAM, ROM, XFER, CALLs 70, 71, 72
 - RS-232, 2-2
 - RS-232/423, definition, Using-3
 - RS-422, definition, Using-3
 - RS-485, definition, Using-3
 - RS-232 interface, 2-9
 - RS-232 network, 12-24
 - RS-422 network, 12-24
 - RS-485 interface, 2-9
 - RS-485 network, 12-24
 - RS-232 interface, 4-4, A-6
 - RS-422, 2-2
 - RS-485, 2-2
 - RUN, 10-19
 - See also* Control C
 - run mode, 4-1
 - running a program, 4-9
 - See also* RUN
- S**
- SA/SB data recorder, SOC-5
 - sample block-transfer BASIC program, 5-7
 - sample ladder logic
 - CALL 118, 13-56
 - CALL 122, 13-65
 - CALL 123, 13-74

- CALL 32, 12-22
- CALL 33, 12-28
- CALL 34, 12-33
- CALL 49, 12-49
- CALL 50, 12-57
- PLC-2 processor, 5-8
- PLC-3 processor, 5-9
- PLC-5 processor, 5-10, 5-11
- PLC-5/250 processor, 5-12
- SCADA, definition, Using-3
- scalar variable, definition, Using-3
- scalar variables, 9-2
- serial ports, SOC-2, 2-1
 - overview, 2-1
- Series A, configuration plug settings, D-1
- set block-transfer-write length, CALL 4, 12-4
- set block-transfer-read length, CALL 5, 12-4
- set breakpoints, 6-4
 - See also* CONT
- set program port communication rate, CALL 78, 13-8
- set wall clock date, CALL 41, 12-40
- set wall clock day of week, CALL 42, 12-40
- set wall clock time, CALL 38, 12-39
- setting, configuration jumpers, 1-3
- setup calls, 7-9
- SGN, functional operator, 9-12
- simple expression, 9-1
- SIN, trigonometric operator, 9-10
- SLC 16-bit binary, 8-5
 - See also* CALLs 14 and 24
- SLC 16-bit signed integer, 8-4
 - See also* CALLs 14 and 24
- SLC 16-bit signed integer to BASIC floating point, CALL 14, 12-8
 - See also* CALL 24
- SLC 16-bit unsigned integer, 8-5
 - See also* CALLs 15 and 25
- SLC 16-bit unsigned integer to BASIC floating point, CALL 15, 12-9
 - See also* CALL 25
- SLC 500, definition, Using-3
- SLC processor
 - read remote data file, CALL 49, 12-44
 - read/write to from BASIC module internal string, CALL 29, 12-18
 - unsolicited write, CALL 118, 13-52
 - write to remote data file, CALL 50, 12-50
- SLC processors, communication with, 2-3
- SNGLSTP, 10-20
 - BASIC command, 6-4
 - See also* CONT
- software features, A-3
- software handshaking, 2-4, 2-8, 11-27, 11-29
- special function operators, 9-17
 - CBY, 9-18
 - DBY, 9-18
 - EOF, 9-17
 - FREE, 9-17
 - LEN, 9-17
 - MTOP, 9-18
 - See also* CALL 77
 - TIME, 9-19
 - See also* CLOCK1, ONTIME
 - XBY, 9-19
- specifications
 - clock/calendar accuracy, A-12
 - communication distances, A-12
 - environmental, A-11
 - port driver and receiver, A-11
 - port isolation, A-12
 - power requirements, A-11

- SQR, functional operator, 9-12
- ST@, 11-35
 - See also* LD@, CALL 77
- statement, IDLE, 11-14
- statements, 4-2, 11-1
 - See also* Chapter 11
 - CLEAR, 11-2
 - CLEARI, 11-3
 - CLEARs, 11-3
 - CLOCK0, 11-4
 - CLOCK1, 11-5
 - DATA, 11-6
 - DIM, 11-7
 - DO-UNTIL, 11-8
 - DO-WHILE, 11-9
 - END, 11-10
 - FOR-TO-(STEP)-NEXT, 11-11
 - GET, 11-12
 - GET#, 11-12
 - GET@, 11-12
 - GOSUB, 11-13
 - GOTO, 11-14
 - IF-THEN-ELSE, 11-15
 - INPL, 11-16
 - INPL#, 11-16
 - INPL@, 11-16
 - INPS, 11-16
 - INPS@, 11-16
 - INPUT, 11-17
 - INPUT#, 11-17
 - INPUT@, 11-17
 - LD@, 11-18
 - LET, 11-19
 - NEXT, 11-21
 - ON-GOSUB, 11-24
 - ONDF1, 11-22
 - ONERR, 11-23
 - ON-GOTO, 11-26
 - ONTIME, 11-25
 - PH0., 11-27
 - PH0#, 11-27
 - PH0@., 11-27
 - PH1., 11-27
 - PH1#, 11-27
 - PH1@., 11-27
 - POP, 11-28
 - PRINT, 11-29
 - PRINT#, 11-29
 - PRINT@, 11-29
 - PUSH, 11-30
 - quick reference, E-1
 - READ, 11-31
 - RESTORE, 11-32
 - RETI, 11-33
 - RETURN, 11-34
 - ST@, 11-35
 - STOP, 6-4, 11-36
 - See also* CONT
 - STRING, 11-37
 - using, 7-1
 - See also* Chapter 11
- status
 - DF1 status, CALL 115, 13-49
 - DH-485 remote read, CALL 87, 13-15
 - DH-485 remote write, CALL 86, 13-14
- status calls, 7-10
- STOP, 11-36
 - execution control and interrupt support function, 6-4
 - See also* CONT
- stopping a program, 4-9
 - See also* JW4, CALLs 18 and 19
- STRING, 11-37
- string
 - append, CALL 61, 12-60
 - conversion from number, CALL 62, 12-61
 - conversion to number, CALL 63, 12-62
 - delete a string from a string, CALL 67, 12-66
 - determine length of, CALL 68, 12-67
 - error messages, C-5

- find a string in a string, CALL 64, 12-63
- insert a string in a string, CALL 66, 12-65
- repeat string, CALL 60, 12-59
- replace a string in a string, CALL 65, 12-64
- STRING # NOT ALLOCATED, C-4
- string calls, 7-5
 - See also* Chapter 8
- string data types, 8-2
- string operators, 9-14
 - ASC, 9-14
 - See also* STRING
 - CHR, 9-16
 - See also* STRING, CALL 65
- string to number conversion, CALL 63, 12-62
- subtract (-), arithmetic operator, 9-5
- summary of changes, SOC-1
- support information, Using-6
- synchronous block-transfer, PLC-5 processor, 5-11
- synchronous block-transfers, 5-3
- system subroutines, 4-2
 - See also* Chapters 12 and 13
- T**
- TAN, trigonometric operator, 9-10
- terms, Using-3
- TIME, special function operator, 9-19
 - See also* CLOCK1, ONTIME
- time
 - retrieve numeric, CALL 46, 12-42
 - retrieve string
 - CALL 43, 12-41
 - CALL 45, 12-42
 - set, CALL 40, 12-39
 - time retrieve numeric, CALL 46, 12-42
 - time retrieve string, CALL 45, 12-42
 - timed-block-transfer-read buffer, CALL 2, 12-2
 - timed-block-transfer-write buffer, CALL 3, 12-3
 - tips, programming, 4-2
 - transfer BASIC output buffer to DH-485 Common Interface File, CALL 85, 13-13
 - transfer data from BTW buffer to PRT1 or PRT2, CALL 34, 12-29
 - transfer data from PRT1 or PRT2 to BTR buffer, CALL 33, 12-23
 - transfer DH-485 common interface file to BASIC input buffer, CALL 84, 13-12
 - transmit buffer
 - See also* output buffer
 - clear PRT1 buffer, CALL 96, 13-33
 - clear PRT2 buffer, CALL 37, 12-35
 - definition, Using-3
 - number of characters in (PRT2), CALL 36, 12-35
 - print PRT1 buffer, CALL 103, 13-36
 - print PRT2 buffer, CALL 110, 13-45
 - PRT1 and PRT2, 2-3
 - retrieve number of characters (PRT1), CALL 95, 13-32
 - software handshaking, 2-4
 - transmit DF1 packet, CALL 114, 13-48
- trigonometric operators, 9-10
 - ATN, 9-11
 - COS, 9-10
 - SIN, 9-10
 - TAN, 9-10
- troubleshooting, C-1

- contacting Allen-Bradley, Using-6
 - error messages from BASIC, C-2
 - error messages from CALL routines, C-4
 - LED indicators, C-1
 - memory support error messages, C-5
 - miscellaneous error messages, C-6
 - PRT2 error messages, C-4
 - string error messages, C-5
 - wall clock error messages, C-4
 - truth table, bitwise operations, 9-7
 - turbo mode, 1-5
 - turbo speed, SOC-3
- U**
- UART, software handshaking, 2-4
 - unpacking BASIC module, 1-2
 - unsupported CALLs, SOC-5
 - upload user EEPROM code to host, CALL 101, 13-35
 - user LED control, CALL 112, 13-47
 - user memory module map, 13-11
 - user PROM check and description, CALL 81, 13-10
 - using the manual, Using-2
- UV PROM**
- See also* memory module definition, Using-3
- V**
- variables
 - arrayed, 9-2
 - dimensioned, 9-2
 - in general, 9-2
 - name of, 9-2
 - protected, CALL 77, 13-6
- See also* LD@, MTOP, ST@
 - scalar, 9-2
 - VER, 10-21
- W**
- wall clock, A-2
 - set date, CALL 41, 12-40
 - set day of week, CALL 42, 12-40
 - set time, CALL 40, 12-39
 - warnings, guarding against ESD, 1-1, 3-1
 - watchdog timer, JW1, 1-4
 - who should use this manual, Using-1
 - write BASIC output buffer to remote DH-485 data file, CALL 91, 13-22
 - write block words, 1-6
 - write output buffer to remote DH-485 common interface file, CALL 93, 13-29
 - write remote DF1 PLC data file, CALL 123, 13-66
 - write to remote DH-485 SLC data file, CALL 50, 12-50
 - write/read to a PLC/SLC from the BASIC module internal string, CALL 29, 12-18
- X**
- XBY, special function operator, 9-19
 - XFER, 10-21
 - See also* PROG, PROG1, PROG2, RAM, ROM
 - XOFF, 2-4, 2-8, 10-6, 11-27, 11-29
 - XON, 2-4, 2-8, 10-6, 11-27, 11-29
 - XOR, 9-8

PLC, PLC-2, PLC-3, PLC-5 are registered trademarks of the Allen-Bradley Company, Inc.
PLC-5/250, SLC, SLC 500 are a trademarks of the Allen-Bradley Company, Inc.
Intel is a trademark of the Intel Corporation.



If you find a problem with our documentation, please complete and return this form.

Pub. Name BASIC Module User Manual

Cat. No. 1771-DB/B Pub. No. 1771-6.5.113 Pub. Date May 1998 Part No. 955127-97

Check Problem(s) Type:	Describe Problem(s):	Internal Use Only
<input type="checkbox"/> Technical Accuracy	<input type="checkbox"/> text <input type="checkbox"/> illustration	
<input type="checkbox"/> Completeness What information is missing?	<input type="checkbox"/> procedure/step <input type="checkbox"/> illustration <input type="checkbox"/> definition <input type="checkbox"/> example <input type="checkbox"/> guideline <input type="checkbox"/> feature <input type="checkbox"/> explanation <input type="checkbox"/> other	<input type="checkbox"/> info in manual (accessibility) <input type="checkbox"/> info not in manual
<input type="checkbox"/> Clarity What is unclear?		
<input type="checkbox"/> Sequence What is not in the right order?		
<input type="checkbox"/> Other Comments Use back for more comments.		

Your Name _____ Location/Phone _____

Return to: Marketing Communications, Rockwell Automation, 1 Allen-Bradley Drive, Mayfield Hts., OH 44124

Phone: (440)646-3176
FAX: (440)646-4320

PLEASE FASTEN HERE (DO NOT STAPLE)

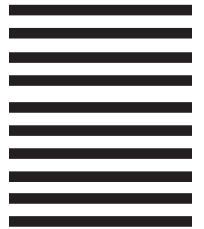
Other Comments

PLEASE FOLD HERE



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

PLEASE REMOVE



BUSINESS REPLY MAIL

First Class Mail

Permit No. 18235

Cleveland, OH

POSTAGE WILL BE PAID BY
ADDRESSEE



Allen-Bradley

1 ALLEN BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705





ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

Allen-Bradley has been helping its customers improve productivity and quality for 90 years. A-B designs, manufactures and supports a broad range of control and automation products worldwide. They include logic processors, power and motion control devices, man-machine interfaces and sensors. Allen-Bradley is a subsidiary of Rockwell International, one of the world's leading technology companies.



With major offices worldwide.

Algeria • Argentina • Australia • Austria • Bahrain • Belgium • Brazil • Bulgaria • Canada • Chile • China, PRC • Colombia • Costa Rica • Croatia • Cyprus • Czech Republic • Denmark • Ecuador • Egypt • El Salvador • Finland • France • Germany • Greece • Guatemala • Honduras • Hong Kong • Hungary • Iceland • India • Indonesia • Israel • Italy • Jamaica • Japan • Jordan • Korea • Kuwait • Lebanon • Malaysia • Mexico • New Zealand • Norway • Oman • Pakistan • Peru • Philippines • Poland • Portugal • Puerto Rico • Qatar • Romania • Russia-CIS • Saudi Arabia • Singapore • Slovakia • Slovenia • South Africa, Republic • Spain • Switzerland • Taiwan • Thailand • The Netherlands • Turkey • United Arab Emirates • United Kingdom • United States • Uruguay • Venezuela • Yugoslavia

World Headquarters, Allen-Bradley, 1201 South Second Street, Milwaukee, WI 53204 USA, Tel: (1) 414 382-2000 Fax: (1) 414 382-4444