Allen-Bradley

BASIC Language

(Catalog Numbers 1746-BAS and 1746-BAS-T)

Reference Manual

Rockwell Automation

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, Rockwell International Corporation does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Rockwell Automation publication SGI-1.1, *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (available from your local Rockwell Automation office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

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 a hazard
- recognize the consequences

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

Chapter 5

Command Line CALLs

Chapter 6

Assignment Functions

Chapter 7

Control Functions

Chapter 8

Execution Control and Interrupt Support Functions

Chapter 9

Math and Backplane Conversion Functions

Chapter 10

Clock/Calendar Functions

Chapter 11

Status Functions

Chapter 12

Output Functions

Chapter 13

Input Functions

Chapter 14

Setup Functions

Chapter 15

String Functions

Read this preface to familiarize yourself with the rest of the manual. This preface covers the following topics:

- who should use this manual
- the purpose of this manual
- how to use this manual
- terms and abbreviations
- conventions used in this manual
- Rockwell Automation support

Who Should Use This Manual

Use this manual if you are responsible for designing, installing, programming, or troubleshooting control systems that use Allen-Bradley small logic controllers.

You should have a basic understanding of SLC 500™ products. You should understand programmable controllers and be able to interpret the ladder logic instructions required to control your application. If you do not, contact your local Rockwell Automation representative for information on available training courses before using this product.

Purpose of this Manual

This manual is a reference guide for programming the BASIC or BASIC-T module. This manual is intended for reference purposes only.

Chapter	Title	Contents
	Preface	Describes the purpose, background, and scope of this manual. Also lists related publications.
1	Language Elements	Describes BASIC program lines, line numbers, statements, commands, operators, and line length.
2	Data Types	Describes and illustrates data types, variable names and types.
3	Expressions and Operators	Describes and illustrates arithmetic, logical, relational, trigonometric, functional, logarithmic, string, and special function operators.
4	BASIC Commands	Describes and illustrates BRKPNT, CONT, [CTRL-C], [CTRL-S], [CTRL-Q], EDIT, ERASE, IDLE, LIST, LIST@, LIST#, MODE, NEW, NULL, PROG, PROG1, PROG2, RAM, REM, REN, ROM, RROM, RUN, SNGLSTP, VER, and XFER commands and CALLs 18 and 19.
5	Command Line CALLS	Describes and illustrates CALLs 73, 74, 77, 81, 82, 101, 103, 104, 109, 110, and 111.
6	Assignment Functions	Describes and illustrates CLEAR, CLEARI, CLEARS, DATA, DIM, LET and RESTORE functions.
7	Control Functions	Describes and illustrates CLOCK1, CLOCK0, DO-WHILE, DO-UNTIL, END, FOR-TO-(STEP)-NEXT, GOTO, IF-THEN-ELSE, NEXT, and ON-GOTO functions.
8	Execution Control and Interrupt Support Functions	Describes and illustrates CALLs 16, 17, 20, 21, 26, 38, 70, 71, 72, and GOSUB, ONERR, ON-GOSUB, ONTIME, PUSH, POP, RETI, RETURN, and STOP functions.
9	Math and Backplane Functions	Describes and illustrates CALLs 14, 15, 24, 25, 88, and 89.
10	Clock/Calendar Functions	Describes and illustrates CALLs 40, 41, 42, 43, 44, 45, 46, 47, 48, and 52.
11	Status Functions	Describes and illustrates CALLs 36, 51, 55, 58, 59, 75, 80, 86, 87, 95, 97, 98, 108, 113, 120, and 121.
12	Output Functions	Describes and illustrates CALLs 23, 28, 29, 31, 37, 54, 57, 85, 91, 93, 94, 96, 112, 114, 115, 123, and PRINT, PH0. PH1. and ST@ functions.
13	Input Functions	Describes and illustrates CALLs 22, 27, 29, 35, 53, 56, 84, 90, 92, 117, 118, 122, and GET, INPL, INPS, INPUT, LD@ and READ functions.
14	Setup Functions	Describes and illustrates CALLs 30, 78, 99, 105, 119, and MODE functions.
15	String Functions	Describes and illustrates CALLs 60, 61, 62, 63, 64, 65, 66, 67, 68, and STRING functions.
Appendix A	Decimal/Hexidecimal/Octal/ ASCII Conversion Table	Lists the Decimal/Hexidecimal/Octal/ASCII equivalents.
Appendix B	BASIC Command, Statement, and CALL Quick Reference Guide	Lists the various commands, statements, and CALLs needed for BASIC programming.

Related Documentation

The following documents contain additional information regarding Rockwell Automation products. To obtain a copy, contact your local Rockwell Automation office or distributor.

For	Read this document	Publication Number
A BASIC and BASIC-T manual that provides information on installing and using the 1746-BAS and 1746-BAS-T modules.	SLC 500™ BASIC and BASIC-T Modules User Manual	1746-UM004A-US-P
A programming manual with detailed instructions on installing and using BASIC Development Software to program the BASIC and BASIC-T modules.	BASIC Development Software Programming Manual	1746-PM001A-US-P
An overview of the SLC 500 family of products	SLC 500™ System Overview	1747-SO001A-US-P
A description of how to install and use a Modular SLC 500 Processor	Modular Hardware Style Installation and Operation Manual	1747-6.2
A reference manual that contains status file data and instruction set information for SLC 500 processors	SLC 500™ and MicroLogix™ 1000 Instruction Set Reference Manual	1747-6.15
A description of how to install and use a module that acts as a bridge between DH485 networks and devices requiring DF1 protocol.	DH-485/RS-232C Interface Module User's Manual	1747-6.12
An application example demonstrating how to transfer ASCII data to an SLC 5/02 or later processor using a remote SLC 500 BASIC module.	ASCII Data Transfer to the SLC 500™ BASIC Module (Series B)	1746-2.41
In-depth information on grounding and wiring Allen-Bradley programmable controllers	Allen-Bradley Programmable Controller Grounding and Wiring Guidelines	1770-4.1
A glossary of industrial automation terms and abbreviations	Allen-Bradley Industrial Automation Glossary	AG-7.1
An article on wire sizes and types for grounding electrical equipment	National Electric Code	Published by the National Fire Protection Association of Boston, MA

How to Use this Manual

To use this manual effectively, use the worksheets provided in Appendix B. The worksheets can help you document your application and settings and also facilitate the flow of information to other individuals in your organization for implementation.

Terms and Abbreviations

The following terms and abbreviations are specific to this product. For a complete listing of Allen-Bradley terminology, refer to the Allen-Bradley Industrial Automation Glossary, publication number ICCG-7.1.

- Module — SLC 500 BASIC and BASIC-T Modules (catalog numbers 1746-BAS and 1746-BAS-T)
- BASIC development software — BASIC Development Software (catalog number 1747-PBASE)
- BASIC —the BASIC-52 programming language
- 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.
- DH485 — network communication protocol
- EPROM — Erasable Programmable Read Only Memory
- EEPROM — Electrically Erasable Programmable Read Only Memory
- memory module — BASIC or BASIC-T modules EEPROM or UVPROM
- MTOP — system control value that holds the last valid memory address
- program port — the port used to program the module. Either PRT1 or port DH485 can be used as the program port.
- RAM — Random Access Memory
- ROM — Read Only Memory, refers to the optional memory module memory space (EEPROM or UVPROM)
- RS-232/423 — serial communication interface
- RS-422 — differential communication interface
- RS-485 — network communication interface
- SCADA — Supervisory Control and Data Acquisition
- scalar variable — a variable with a single value
- SLC 500 — SLC 500 fixed and modular controller
- UVPROM — Ultra Violet Erasable Programmable Read Only Memory

Conventions Used in this Manual

The following conventions are used throughout this manual:

- Bulleted lists such as this one provide information, not procedural steps.
- Numbered lists provide sequential steps or hierarchical information.
- *Italic* type is used for emphasis.
- Text in **`this font`** indicates words or phrases you should type.
- Key names match the names shown and appear in bold, capital letters within brackets (for example, **`[ENTER]`**).

Rockwell Automation 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, plus Rockwell Automation representatives in every major country in the world.

Local Product Support

Contact your local Rockwell Automation representative for:

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

Technical Product Assistance

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

Your Questions or Comments on this Manual

If you find a problem with this manual, please notify us of it on the enclosed Publication Problem Report.

If you have any suggestions for how this manual could be made more useful to you, please contact us at the address below:

Rockwell Automation
Control and Information Group
Technical Communication, Dept. A602V
P.O. Box 2086
Milwaukee, WI 53201-2086

Language Elements

This chapter introduces you to the elements of a BASIC program. These elements include BASIC:

- line numbers
- statements, commands, and operators
- line length

Character Set

BASIC programs are composed of a group of BASIC program lines. Each BASIC program line is composed of a group of ASCII characters. Refer to Appendix A for a complete listing of ASCII character codes.

The BASIC Program Line

BASIC program lines consist of a BASIC line number and BASIC statements and operators. BASIC program lines are restricted to the BASIC line length.

BASIC Line Numbers

We refer to BASIC line numbers as:

[ln num]

BASIC line numbers indicate the order that the program lines are stored in memory and are also used as references when branching and editing. This number may be any whole integer from 1 to 65535. 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.

Since the computer runs the statements in numerical order, additional lines need not appear in consecutive order on the screen. 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 reentering an entire program if you forget to include a line.

IMPORTANT The first line of your program must be a comment.

Typically, the line numbers of a program start out looking like the first column and end up looking something like the second column below:

```
#1          #2
10           5
20           7
30          10
40          15
50          20
60          30
70          35
80          40
 .           .
 .           .
 .           .
```

IMPORTANT Reuse of an existing line number causes all of the information referenced by the original line number to be lost. Be careful when entering numbers in the Command mode, as you may accidentally erase some program lines. You may delete an existing line by retyping it with no information following it and pressing **[RETURN]**.

BASIC Statements, Commands, and Operators

BASIC program lines consist of a BASIC line number and BASIC statements and operators. Depending on the logic of your program, there may be more than one statement on a line. If so, each must be separated by a colon (:).

BASIC Line Length

A BASIC program line always begins with a line number and must contain at least one character, but no more than 68 characters. A program line ends when you press **[RETURN]**.

Data Types

This chapter provides you a method of defining or displaying data within the BASIC programming language through the use of:

- data types
- variables

Data Types

Data types are broken down into three sections: argument stack, string and numeric elementary data types, and backplane conversion data.

Argument Stack

The argument stack (A-stack) stores all constants that the BASIC or BASIC-T module is currently using. Operations 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. 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 saves data to the argument stack and the POP command restores data from the stack. PUSHes and POPs are typically associated with CALLs. PUSHes and POPs are mechanisms used to transfer information to and from CALL routines.

PUSH makes a copy of the variable being 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 being POPped.

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 module uses single-dimension string variables, **`$([expr])`**. The dimension of a string variable (the **`[expr]`** value) ranges from 0 to 254. This means that you can define and manipulate 255 different strings in the module. Initially, no memory is allocated for strings. Memory is allocated using the STRING statement. Strings are declared and manipulated through the $ operator.

When allocating memory for a string, you must account for the overhead bytes used by BASIC to manipulate strings. BASIC uses one overhead byte per string being declared plus one additional overhead byte.

Example 1

String 106,20

Allocates space for five 20 byte strings (100 bytes) and includes five overhead bytes (1 per string) and one additional overhead byte.

In the module you can define strings with the LET statement, the INPUT statement, and with the ASC operator.

Example 2

```
>10 STRING 106,20
>20 $(1)="THIS IS A STRING, "
>30 INPUT "WHAT'S YOUR NAME? - ",$(2)
>40 PRINT $(1),$(2)
>50 END

READY
>RUN

WHAT'S YOUR NAME? - FRED
THIS IS A STRING, FRED

READY
>
```

You can also assign strings to each other with a LET statement.

Example 3

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

Result: Assigns the string value in $(1) to the STRING $(2).

Numeric Data Types

There are two different numeric data types:

- integer numbers
- floating-point numbers

You can enter and display numbers in four formats: integer, decimal, hexadecimal, and exponential.

Example

```
129, 34.98, 0A6EH, 1.23456E+3
```

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

Integer Numbers

The module operates on unsigned 16-bit integers that range 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 (example: 170H). If the hexadecimal number begins with A through F, then it must be preceded by a zero. (For example, you must enter A567H as 0A567H.) When an operator, such as .AND. requires an integer, the module truncates the fraction portion of the number so it fits the integer format. Integers are shown as:

[integer]

IMPORTANT The SLC 500 processor operates on signed 16-bit integers that range from –32768 to 32767. If an integer value larger than 32767 is passed to the processor from the module, that value is interpreted as negative by the processor.

Floating-Point Numbers

In the module, all numbers are stored 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 processor accounts for the location of the decimal point. This allows the processor to store only the significant digits of a value, thus saving memory space.

You can represent the following range of numbers in the module:

+1E –127 to +.99999999 +127

There are eight significant digits. Numbers are internally rounded to fit this precision.

Backplane Conversion Data

The module communicates with the local processor through the SLC 500 I/O backplane. All data communicated to and from the SLC 500 is in SLC 500 format. The SLC 500 formats are:

- 16-bit signed integer (–32768 to 32767)
- 16-bit binary (0000000000000000 to 1111111111111111)

IMPORTANT Any integer larger than 32767 is interpreted as a negative number by the SLC 500 processor

Variables

Variables that include a single-dimension expression [exp] are dimensioned or arrayed variables. Variables that contain a letter or a letter and a number are scalar variables. Any variables entered in lower case are changed to upper case. Variables are shown as:

[var]

The module allocates variables in a static manner, which means the first time a variable is used, the module allocates a portion of memory (8 bytes) specifically for that variable. This memory cannot be de-allocated on a variable to variable basis. This means that if you execute a statement (example: >10 Q - 3), you cannot tell the module that the variable Q no longer exists to free up the 8 bytes of memory that belong to Q. You can clear the memory allocated to variables by executing a CLEAR statement. The CLEAR statement frees all memory allocated to variables. Variables may be set aside for reuse to save memory.

IMPORTANT The module requires less time to find a scalar variable because there is no expression to evaluate. To run a program as fast as possible, use single-dimension 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.

Variable Names

Variables may represent either numeric values or strings. Variable names can only be eight characters long. The module compares the first, last, and number of characters in a variable name with the first, last, and number of characters in other variable names to determine if it is a unique variable name. The characters allowed in a variable name are letters, numbers, and the decimal point. Special type declaration characters are also allowed.

A variable can be a letter (for example **A**, **X**, or **I**) followed by a:

- single-dimension expression, (example: **J(4), G(A+6), I(10*SIN(X))**)
- number followed by a single-dimension expression (example: **A1(8), P7(10*SIN(X)), W8(A + C)**)
- number (0 to 9) or letter (example: **AA, AC, XX, A1, X3, G8**) except for the following combinations: **CR, DO, IE, IF, IP, ON, PI, SP, TO, UI, UO**

IMPORTANT Reserved words (words already used in BASIC functions or statements) cannot be used as variable names.

Variable Types

Type declaration characters indicate what a variable represents. The following type declaration character is recognized:

Character	Variable Type
$	String variable

The only other legal variable type is a floating-point variable. Floating-point variables do not require a type declaration.

Expressions and Operators

This chapter describes and illustrates how you manipulate and/or evaluate expressions and statements within the BASIC program or the command line. Table 3.1 lists the corresponding mnemonics.

Table 3.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Absolute value	ABS( )	3-9
Return the integer value of the ASCII character.	ASC( )	3-12
Return the arctangent of the argument.	ATN( )	3-8
Retrieve data from the specified memory address.	CBY( )	3-16
Count the value converted ASCII character.	CHR( )	3-14
Return the cosine of argument.	COS( )	3-8
Retrieve or assign data to or from the internal data memory of the BASIC or BASIC-T module.	DBY( )	3-16
Test for empty input buffer.	EOF	3-15
"e" (2.7182818) TO THE X	EXP( )	3-11
Test for number of free bytes of RAM memory.	FREE	3-15
Integer	INT( )	3-10
Read the number of bytes of memory in the current selected program.	LEN	3-15
Natural log	LOG( )	3-11
Read the last valid memory address.	MTOP	3-16
One's complement	NOT( )	3-9
PI-3.1415926	PI	3-10
Random number	RND	3-11
Sign	SGN	3-10
Return the sine of the argument	SIN( )	3-8
Square Root	SQR( )	3-10
Return the tangent of the argument.	TAN( )	3-8
Retrieve and/or assign the free running clock value.	TIME	3-17
Retrieve or assign data to or from the external data memory of the module.	XBY( )	3-17
Addition	+	3-3
Division	/	3-4
Exponentiation	**	3-4
Multiplication	*	3-4
Subtraction	-	3-4
Logical AND	.AND.	3-6
Logical OR	.OR.	3-6

Table 3.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Logical Exclusive OR	.XOR.	3-6
Direct communications to port PRT1.	@	3-15
Direct communications to port PRT2.	#	3-15

Expressions and Operators

An expression is a logical mathematical expression that involves operators, constants, and variables. There are eight types of operators that may act on an expression:

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

Expressions

Expressions are simple or complex.

Simple expression: **`12*EXP(A)/100,H(1) + 55,`**

Complex expression: **`(SIN(A)*SIN(A)+COS(A)* COS(A)/2)`**

A stand alone variable [var] or constant [const] is also considered an expression. Expressions are shown as:

[expr]

Operators

An operator performs a defined operation on variables or constants. Operators require either one or two operands. Typical two operand operators include ADD(+), SUBTRACT(-), MULTIPLY(*) and DIVIDE(/). We call operators that require only one operand, single-operand operators. Typical single-operand operators are SIN, COS, and ABS.

Hierarchy of Operators

The hierarchy of operators is the order that the operations in an expression are performed. You can write complex expressions using only a small number of parentheses. To illustrate the hierarchy of operators, examine the following equation:

4+3*2 = ?

In this equation, multiplication has precedence over addition. Therefore, multiply (3*2) and then add 4.

4+3*2 = 10

When an expression is scanned from left to right, an operation is not performed until an operator of lower or equal precedence is encountered. In the example, you cannot perform addition until the multiplication operation is complete because multiplication has a higher precedence. Use parentheses if you are in doubt about the order of precedence or to enhance program readability. The precedence of operators from highest to lowest in the module is:

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.)

Arithmetic Operators

The module contains a complete set of arithmetic operators that are divided into two groups: dual-operand operators and single-operand operators.

The general form of all dual-operand instructions is:

(expr) OP (expr), where OP is one of the following arithmetic operators

Add (+)

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

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

Divide (/)

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

Example	Result
`>PRINT 100/5`	`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.

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

Multiply (*)

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

Example	Result
`>PRINT 3*3`	`9`

Subtract (-)

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

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

Negation (-)

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

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

Overflow and Division by Zero

During the evaluation of an expression if an overflow, underflow, or division by zero error occurs, the module generates error messages and reverts to Command mode. Refer to the ONERR operation in chapter 8 for more information on how to trap these errors.

The largest result allowed from any calculation is 0.99999999 E+127. If this number is exceeded, the module generates the **ERROR: ARITH. OVERFLOW** message and returns to Command mode.

The smallest result allowed from any calculation is 0.99999999 E-128. If this number is exceeded, the module generates the **ERROR: ARITH. UNDERFLOW** message and returns to Command mode.

If an attempt is made to divide any number by zero, the module generates the **ERROR: DIVIDE BY ZERO** message and returns to Command mode.

```
>10 PRINT 9/0
>20 PRINT “PROGRAM SHOULD NOT GET HERE.”

READY
>RUN

ERROR: DIVIDE BY ZERO - IN LINE 10

10     PRINT 9/0
----------------X

READY
>

>10 PRINT 9.9E126*(2)
>

READY
>RUN

ERROR: ARITH. OVERFLOW - IN LINE 10

10     PRINT 9.9E126*(2)
------------------------X

READY
>
```

Logical Operators

The module contains a complete set of logical operators that are divided into two groups: dual-operand operators and single-operand operators.

The general form of all dual-operand instructions is:

`(expr) OP (expr)`, where OP is one of the following logical operators.

These operators perform BIT-WISE logical operations on numbers between 0 (0000H) and 65535 (0FFFFH) inclusive. If the argument is outside this range, then the module generates an **`ERROR: BAD ARGUMENT`** message and returns to Command mode. All decimal places are truncated, not rounded. Use the following table for bit manipulations on 16-bit values.

Table 3.2 Bit Manipulations on 16-Bit Values

X	Y	X .AND.Y	X .OR.Y	X .XOR.Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

.AND.

Use the logical .AND. operator to logically AND expressions together.

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

.OR.

Use the logical .OR. operator to logically OR expressions together.

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

.XOR.

Use the logical exclusive .XOR. operator to logically XOR expressions together.

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

Relational Operators

Relational expressions involve the operators =, < >, >, >=, <, and <=. In the module, relational operations are typically used to test a condition. The 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.

Relational expressions are shown as:

[rel expr]

Example	Result
`>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.. 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([expr]) operator.

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

By chaining relational expressions with logical operators, you can test particular conditions with one statement.

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 module contains a complete set of trigonometric operators. These operators are single-operand operators.

SIN([expr])

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.

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

COS([expr])

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.

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

TAN([expr])

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

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

ATN([expr])

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 –PI/2 (3.1415926/2) and PI/2.

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

Comments on Trigonometric Functions

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 PI/2. This reduction is accomplished by the following equation:

```
reduced argument=(user arg/PI – INT(user arg/PI) *PI
```

The reduced argument, from the above equation, is between 0 and PI. The reduced argument is then tested to see if is greater than 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 (example: **`greater than 1000`**). This is because significant digits in the decimal (fraction) portion of the reduced argument are lost in the **`(user arg/PI - INT (user arg/PI))`** expression. As a general rule, keep the arguments for the trigonometric functions as small as possible.

Functional Operators

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

ABS([expr])

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

Example	Result
`>PRINT ABS(5)`	`5`
`>PRINT ABS(–5)`	`5`

NOT([expr])

Use the NOT operator to return a 16-bit one's complement of the expression. The expression must be a valid integer (example: **`between 0 and 65535 (0FFFFH) inclusive`**). Non-integers are truncated, not rounded.

Example	Result
`>PRINT NOT(65000)`	`535`
`>PRINT NOT(0)`	`65535`

INT([expr])

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

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

PI

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

SGN([expr])

Use the SGN operator to return a value of +1 if the argument is greater than zero, zero if the argument is equal to zero, and –1 if the argument is less than zero.

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

SQR([expr])

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

Example	Result
`>PRINT SQR(9)`	`3`
`>PRINT SQR(45)`	`6.7082035`
`>PRINT SQR(100)`	`0`

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 and generates 65536 pseudo-random numbers before repeating the sequence. The numbers generated are specifically between 0/65535 and 65535/65535 inclusive.

IMPORTANT Unlike most BASIC languages, the RND operator in the module does not require an argument or a dummy argument. If an argument is placed after the RND operator, a bad syntax error occurs.

Example	Result
`>PRINT RND`	`.26771954`

Logarithmic Operators

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

LOG([expr])

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.

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

If base 10 logs are needed, the following expression may be used:

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

The log is natural.

EXP([expr])

Use the EXP operator to raise the number e (2.7182818) to the power of the argument.

Example	Result
`>PRINT EXP(1)`	`2.7182818`
`>PRINT EXP(LOG(2))`	`2`

String Operators

Two operators in the module can manipulate STRINGS. These operators are ASC( ) and CHR( ).

ASC([expr])

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

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


READY
>RUN

65
65

READY
>
```

The decimal representation for the ASCII character A is 65. The decimal representation for the ASCII character a is 97. However, the module capitalizes all ASCII characters not contained within quotation marks. Similarly, special ASCII characters whose decimal value is greater than 127 should not be used.

In addition, you can evaluate individual characters in a defined ASCII string with the ASC operator.

```
>1  REM EXAMPLE PROGRAM
>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
>
```

When you use the ASC operator as shown above, the $([expr]) denotes what string is accessed. The expression after the comma selects an individual character in the string. In the above 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.

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1)="ABCDEFGHIKJL"
>20 FOR X = 1 TO 12
>30 PRINT ASC($(1),X),
>40 NEXT X
>50 END


READY
>RUN

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

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

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. A simple program can determine if two strings are identical.

```
>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([expr])

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

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

Like the ASC operator, the CHR operator also selects individual characters in a defined ASCII string.

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) = “The module”
>20 FOR I = 1 TO 16 : PRINT CHR($(1),I),: NEXT I


READY
>RUN

The module
READY
>
```

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

Unlike the ASC operator, 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, causing the module to go into Command mode. Use the ASC operator to change a value in a string, or use the string support CALL routine – replace string in a string.

IMPORTANT Use the CHR function only in a print statement. You cannot use the CHR operator in a DATA statement.

Special Function Operators

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

and @

Use the # and @ operators to direct communications. Communication takes place through port PRT1 when the @ operator is programmed, and through port PRT2 when the # operator is programmed. The absence of either the # or @ operators indicates that communication should take place through a program port (port PRT1 or port DH485).

Example	Result
`>10 A = GET#`	The next character in the PRT2 input buffer is assigned to variable A.
`>10 A = GET@`	The next character in the PRT1 input buffer is assigned to variable A.

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 the user. When the current selected program is in RAM, the following relationship is true:

```
FREE = MTOP – LEN – 511
```

LEN

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

IMPORTANT The module does not require any dummy arguments for the system control values.

MTOP

Use the MTOP operator to retrieve the last valid memory address in RAM that is available to the module. After reset, the 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 this value has not been changed by CALL 77, then the last valid BASIC address is 5FFFH (24575).

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

CBY([expr])

Use the CBY operator to retrieve data from the program or code memory address location of the module. You cannot assign CBY a value; it can only be read. 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.

IMPORTANT Improper use of this operator may cause a malfunction of the module.

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

DBY([expr])

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

IMPORTANT Improper use of this operator may cause a malfunction of the module.

Example	Result
`A = DBY(B)`	The value in internal memory location B is assigned to variable A. B must be between 0 and 255.
`DBY(250) = CBY(1000)`	The value in program or code memory location 1000 is assigned to internal memory location 250.

XBY([expr])

Use the XBY operator to retrieve or assign data to or from the external data memory of the 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 between 0 and 255 inclusive. If it is not, a bad argument error occurs.

IMPORTANT Improper use of this operator may cause a malfunction of the module.

Example	Result
`A = XBY(0F000H)`	The value in external memory location 0F00H is assigned to variable A.
`XBY(4000H) = DBY(100)`	The value in internal memory location 100 is assigned to external memory location 4000H.

TIME

Use the TIME operator to retrieve or assign a value to the free running clock resident in the module. After reset, time is equal to 0. The CLOCK1 statement enables the free running clock. When the free running clock is enabled, the special function operator TIME increments once every 5 milliseconds. The units of time are in seconds.

When TIME is assigned a value with a LET statement (example: TIME=100), only the integer portion of TIME is changed.

```
>CLOCK1       (enable FREE RUNNING CLOCK)

>CLOCK0       (disable FREE RUNNING CLOCK)

>PRINT TIME   (display TIME)
3.315

>TIME = 0     (set TIME = 0)

>PRINT TIME   (display TIME)
.315          (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) statement. Each count in internal memory location 71 (47H) represents 5 milliseconds of TIME.

Continuing with the example:

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

>PRINT TIME
0

>DBY(71) = 3 (fraction of TIME = 3*5ms = 15 ms)

>PRINT TIME
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, if you want to create an accurate 12-hour clock: there are 43200 seconds in a 12-hour period, so an ONTIME 43200, [ln num] statement is used. 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.

BASIC Commands

This chapter describes and illustrates words and expressions that cause a function to occur within the BASIC program or the command line. Table 4.1 lists the corresponding mnemonics.

Table 4.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Set the program break point.	BRKPNT	4-2
Re-enable the **[CTRL-C]** break function.	CALL 18	4-5
Disable the **[CTRL-C]** break function.	CALL 19	4-6
Continue after a Stop or **[CTRL-C]**.	CONT	4-3
Stop execution & return to Command mode.	CONTROL-C	4-4
Restart a list after **[CTRL-S]**.	CONTROL-Q	4-8
Interrupt a list command.	CONTROL-S	4-7
Edit a line of the BASIC program.	EDIT	4-8
Delete the last BASIC program stored in ROM by a PROG command.	ERASE	4-9
Force the module to enter "wait until interrupt mode".	IDLE	4-10
List the program to the console devise.	LIST	4-11
List the program to the serial printer.	LIST#	4-12
List the program to the device connected to port PRT1.	LIST@	4-12
Set port parameters of ports PRT1, PRT2, and DH485.	MODE	4-12
Erase the program stored in RAM.	NEW	4-14
Set NULL count after carriage return-line feed.	NULL	4-14
Save the current program in EPROM.	PROG	4-15
Save the baud rate information in EPROM.	PROG1	4-16
Save the baud rate information in EPROM and execute the program after reset.	PROG2	4-17
Evoke RAM mode.	RAM	4-19
Specify a remark or comment line.	REM	4-19
Renumber the BASIC program.	REN	4-20
Select ROM mode.	ROM	4-20
Select ROM mode and execute the selected program.	RROM	4-21
Execute a program.	RUN	4-22
Initiate single-step program execution.	SNGLSTP	4-23
Verify the module firmware version.	VER	4-25
Transfer a program from ROM to RAM.	XFER	4-26

BRKPNT

Purpose

Use the BRKPNT command to set a program break point at the line number specified by this command. Program execution stops just before the line number specified by the BRKPNT command. If the line number is zero, the break point is disabled. After the break point is reached, you can examine variables by using assignment statements. Continue from the break point by using the CONT command. 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[ln num]

Example

```
>1   REM EXAMPLE PROGRAM
>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
>PRINT D,SU,AV
 .86042573 0 0

>D = .5

>PRINT D,SU,AV
 .5 0 0

>CONT

THE AVERAGE VALUE IS .48060383

READY
>
```

CONT

Purpose

Use the CONT command to resume execution of a program stopped by a **[CTRL-C]**, BRKPNT command, or a STOP statement. If you stop a program by pressing **[CTRL-C]** on the console device or by execution of a STOP statement, you can resume execution of the program by typing CONT. If you press **[CTRL-C]** while **[CTRL-C]** is enabled, it stops the program. Use CONT to continue. Between the stopping and the re-starting of the program you may display the values of variables or change the values of variables. However, you cannot CONTinue if the program is modified during the STOP or after an error.

IMPORTANT **[CTRL-C]** clears all input and output buffers.

Syntax

CONT

Example

```
>NEW

>1  REM EXAMPLE PROGRAM
>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 [CTRL-C] pressed


STOP - IN LINE 15
READY
>CONT

20
21
22
```

Control-C

Purpose

Use the **[CTRL-C]** command to stop execution of the current program and return the module to the Command mode. In some cases you can continue execution of the program using a CONTinue. See the explanation for CONTinue for more information.

IMPORTANT **[CTRL-C]** clears all input and output buffers.

Syntax

```
[CTRL-C]
```

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          [CTRL-C] pressed

STOP – IN LINE 20

READY
>PRINT I
  27

>I =10

>CONT

  10
  11
  12
```

Notice that after **[CTRL-C]** is pressed and I is printed the value of I is 27. The value of I is incremented several times before **[CTRL-C]** is detected.

CALL 18 – Re-enable the Control-C Break Function

Purpose

Re-enable the **[CTRL-C]** break function by executing CALL 18 in a module program or from the Command mode.

IMPORTANT When **[CTRL-C]** is disabled, you are unable to stop program execution through a BASIC command. Cycling power re-enables **[CTRL-C]** checking until the program once again disables **[CTRL-C]**. To stop program execution, you must cycle power and press **[CTRL-C]** before the line that disables **[CTRL-C]** is executed.

Syntax

CALL 18

Example

```
>1  REM EXAMPLE PROGRAM
>10 CALL 19
.
>90 CALL 18
```

CALL 19 – Disable the Control-C Break Function

Purpose

Disable the **[CTRL-C]** break function by executing CALL 19 in a module program or from the Command mode.

When CALL 19 is executed, the **[CTRL-C]** break function for both LIST and RUN operations is disabled. Cycling power returns the **[CTRL-C]** function to normal operation if it is disabled from the Command mode

.

IMPORTANT **[CTRL-C]** is enabled by default.

Syntax

CALL 19

Example

```
>1  REM EXAMPLE PROGRAM
>10 CALL 19
```

Control-S

Purpose

Use the **[CTRL-S]** command to interrupt the scrolling of a BASIC program during the execution of a LIST command. **[CTRL-S]** stops output from the transmitting port if you are running a program. In this case XOFF (**[CTRL-S]**) operates as follows:

- XOFF only operates on PRINT statements.
- When received during a PRINT, data output is suspended immediately but program execution continues.
- 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.
- XON (**[CTRL-Q]**) is required to resume data output operation.

IMPORTANT **[CTRL-S]** only works if you have enabled software handshaking on the program port.

Syntax

```
[CTRL–S]
```

Example

```
> LIST
1  REM EXAMPLE PROGRAM
10 A = 1
20 DO
[CTRL–S]
.
.
.
[CTRL–Q]
30 A = A+1
40 PRINT A
50 WHILE A < 20

READY
>
```

In this example, the output is suspended when **[CTRL–S]** is pressed. The output is continued after **[CTRL–Q]** is pressed.

Control-Q

Purpose

Use the **[CTRL-Q]** command to restart a LIST command or PRINT output that is interrupted by **[CTRL-S]**.

Syntax

```
[CTRL–Q]
```

Example

```
> LIST
1  REM EXAMPLE PROGRAM
10 A = 1
20 DO
[CTRL–S]
.
.
.
[CTRL–Q]
30 A = A+1
40 PRINT A
50 WHILE A < 20

READY
>
```

In this example, the output is suspended when **[CTRL–S]** is pressed. The output is continued after **[CTRL–Q]** is pressed.

EDIT

Purpose

Use the EDIT command to access the BASIC line editor. Use this editor to edit a line of the current program in RAM. Table 4.2 lists the BASIC editor operations.

Table 4.2 BASIC Editor Operations

Operation	Function	Key Strokes
Move	Use the Move operation to provide right/left cursor control.	**[Space bar]** – moves the cursor one space to the right. **[Backspace]** – 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.
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 **[Ctrl–A]**. Total line length is 79 characters.	**[Ctrl–A]** Important: You must press a second **[Ctrl–A]** to terminate the Insert command.
Delete	Use the Delete operation to delete the character at the cursor position.	**[Ctrl–D]**
Exit	Use the Exit operation(s) to exit the editor with or without saving the changes.	**[Ctrl–Q]** – exits the editor and replaces the old line with the edited line. **[Ctrl–C]** – exits the editor without saving any changes made to the line.
Retype	Use the Retype operation to copy the current line of text and insert it at the line following the current line. The cursor is moved to the first character on the new line.	**[RETURN]**

Syntax

EDIT

Example

```
>EDIT 150
```

Displays program line number 150 for editing.

ERASE

Purpose

Use the ERASE command to delete the last BASIC program stored in EEPROM through a PROG command.

Syntax

ERASE

Example

```
>ERASE

>ROM 13 ERASED
```

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

IDLE

Purpose

Use the IDLE command to force the module to enter wait until Interrupt mode. Program execution halts until an ONTIME condition is met. The ONTIME interrupt must be enabled before executing the IDLE command or else the module enters a wait forever mode. **[CTRL-C]** exits the IDLE command if **[CTRL-C]** is enabled.

Syntax

```
IDLE
```

Example

```
>1  REM EXAMPLE PROGRAM
>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!!!
```

LIST

Purpose

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 module programs. You can terminate the listing of a program at any time by pressing **[CTRL-C]** on the console device. You can interrupt and continue the listing using **[CTRL-S]** and **[CTRL-Q]**.

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 in enabled.

Syntax

LIST [ln num]

LIST [ln num] - [ln num]

The first variation causes the program to print from the designated line number [ln num] to the end of the program. The second variation causes the program to print from the first designated line number [ln num] to the second designated line number [ln num].

IMPORTANT You must separate the two line numbers with a dash (–).

Example

```
>LIST
1  REM EXAMPLE PROGRAM
10 PRINT “LOOP PROGRAM”
20 FOR I = 1 TO 3
30 PRINT I
40 NEXT I
50 END

READY
>LIST 30
1  REM EXAMPLE PROGRAM
30 PRINT I
40 NEXT I
50 END

READY
>LIST 20–40
1  REM EXAMPLE PROGRAM
20 FOR I = 1 TO 3
30 PRINT I
40 NEXT I
```

LIST@

Purpose

Use the LIST@ command to print the program to the device attached to port PRT1. All comments that apply to the LIST command apply to the LIST@ command. You must configure PRT1 port parameters to match your particular list device. The PRT1 parameters (baud rate, parity, stop bits, and so on) can be set using the MODE command.

Syntax

LIST@

Examples

```
LIST@ :REM LISTS ALL LINES IN THE PROGRAM

LIST@10–20 :REM LISTS LINES 10 THROUGH 20
```

LIST#

Purpose

Use the LIST# command to print the program to the device attached to port PRT2. All comments that apply to the LIST command apply to the LIST# command. You must configure the PRT2 port parameters to match your particular list device. The PRT2 parameters (baud rate, parity, stop bits, and so on) can be set using the MODE command.

Syntax

LIST#

Example

Refer to LIST@ examples.

MODE

Purpose

Use the MODE command to set the port parameters of ports PRT1, PRT2, and DH485. Table 4.3 lists the port parameters and default settings for ports PRT1 and PRT2. Table 4.4 lists the port parameters for port DH485.

Table 4.3 PRT1 and PRT2 Port Parameters

Port Parameters	Selections	Default Settings
baud rate	300, 600, 1200, 2400, 4800, 9600, 19200	1200
arg1 (parity)	None (N), Even (E), Odd (O)	N
arg2 (number of data bits)	7 or 8	8
arg3 (number of stop bits)	1 or 2	1
arg4 (handshaking)	No handshaking (N) Software handshaking (S) Hardware handshaking (H) Hardware and software handshaking (B)	S
arg5 (storage type)	Store information in user ROM and RAM (E) Store information in battery backed RAM (R)	R

IMPORTANT If any argument (other than port name and baud rate) is left blank, then that argument defaults to the previously specified value for that argument.

Table 4.4 DH485 Port Parameters

Port Parameters	Selections	Default Settings
baud rate	300, 600, 1200, 2400, 4800, 9600, 19200	19200
arg1 (host node address)	0 to 31	0
arg2 (module node address)	0 to 31	1
arg3 (maximum node address)	1 to 31	31
arg4 (not used)		
arg5 (storage type)	Store information in user ROM and RAM (E) Store information in battery backed RAM (R)	R

IMPORTANT If any argument (other than port name) is left blank, then that argument defaults to the previously specified value for that argument.

Syntax

MODE(port name, baud 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,,,)
```

IMPORTANT The E storage type option cannot be used if MODE is used as a statement.

NEW

Purpose

Use the NEW command to delete the program 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. The NEW command is used to erase a program and all variables in RAM.

Syntax

NEW

Example

```
>NEW

>LIST

>READY
>
```

NULL

Purpose

Use the NULL command to determine how many NULL characters (00H) the module outputs after a carriage return in a print statement. 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]

PROG

IMPORTANT Before you attempt to program EEPROM, read the PROG, PROG1 and PROG2 sections of this chapter.

Purpose

Use the PROG command to program the resident EEPROM with the current program in RAM. The module cannot program UVPROMs.

IMPORTANT Be sure you have selected the program you want to save before using the PROG command. Your 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.

After you type **`PROG`**, the module displays the number in the EEPROM FILE the program occupies. Programming takes only a few seconds.

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 successful.

READY
>ROM 12

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

READY
>
```

The program just placed in the EEPROM is the 12th program stored.

IMPORTANT If you exceed the available EEPROM space, you cannot continue programming until it is erased. Use the ERASE command to erase the last program stored in EEPROM. Be sure to use CALL 81 or CALL 82 to determine memory space prior to programming your EEPROM. Refer to chapter 5 for information regarding CALLs 81 and 82.

PROG1

IMPORTANT Before you attempt to program an EEPROM, read the PROG, PROG1 and PROG2 sections of this chapter.

Purpose

Use the PROG1 command to program the resident EEPROM with port information for all three ports as well as store MTOP information. At module powerup, the 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 baud 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.

Syntax

PROG1

Example

```
READY
>PROG1

Programming sequence successful.
```

PROG2

IMPORTANT Before you attempt to program an EEPROM, read the PROG, PROG1 and PROG2 sections of this chapter. Note, the PROG2 command does not transfer the RAM program to EEPROM. The PROG2 command enables the first program in EEPROM to be loaded at each powerup.

Purpose

Use the PROG2 command the same as you would the PROG1 command except for the following: instead of signing on and entering the Command mode, the module immediately begins executing the first program stored in the resident EEPROM. Otherwise, it executes the program stored in RAM.

You can use the PROG2 command to RUN a program on powerup without connecting to a console. Saving PROG2 information is the same as typing a ROM 1, RUN command sequence. This feature also allows you to write a special initialization sequence in BASIC and generate a custom sign-on message for specific applications. The PROG2 command does not alter the first program in the memory module.

IMPORTANT The PROG2 command does not cause the module to RUN at powerup if PRT1 default communications are selected via JW4. Refer to Chapter 3 of the *SLC 500™ BASIC and BASIC-T Modules User Manual* (publication number 1746-UM004A-US-P) for more information.

The following figure shows the module operation from a power-up condition using PROG1 and PROG2, or battery backed RAM.

Syntax

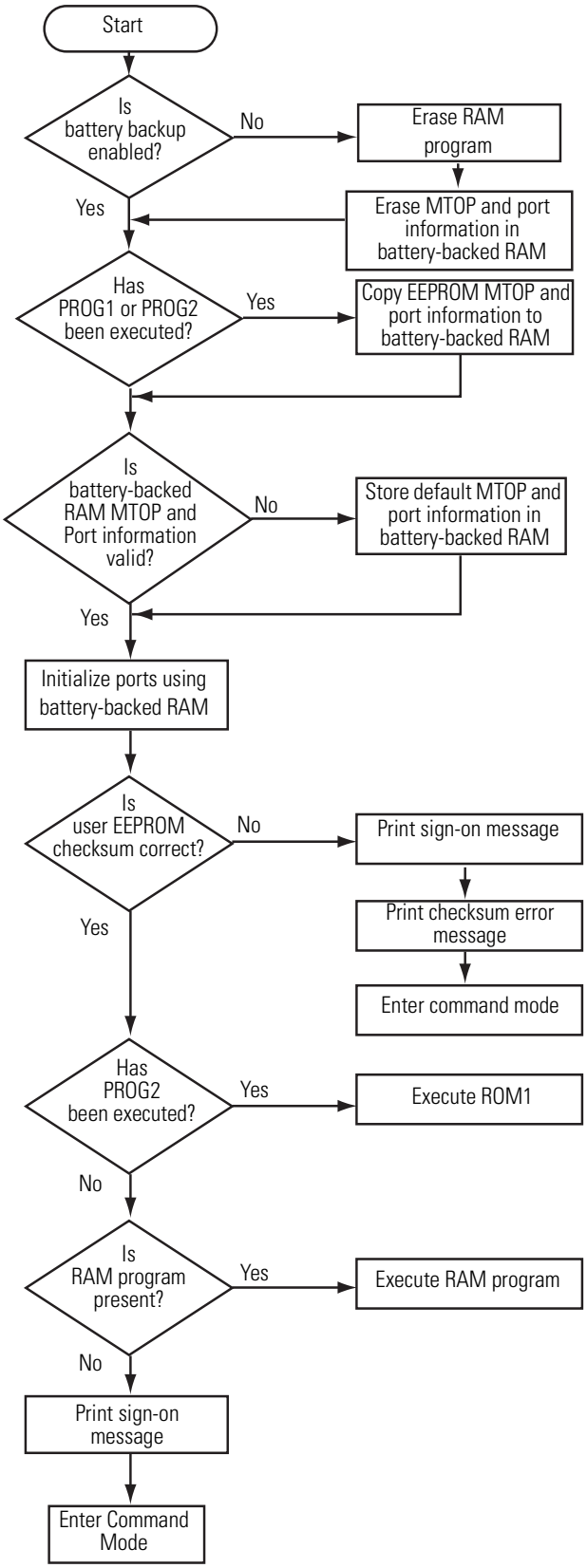
PROG2

Example

```
READY
>PROG2

Programming sequence successful.
```

Figure 4.1 Operation of PROG1 or PROG2



RAM

Purpose

Use the RAM command to tell the module interpreter to select the current program out of RAM. The current program is displayed during a LIST command and executed when **RUN** is typed.

IMPORTANT RAM space is limited to 24K bytes. Use the following formula to calculate the available user RAM space

Available user RAM = MTOP–H

```
H  = LEN + S + 6*A +  8*V + 512
```

Where:

```
LEN  =  system control value that contains current RAM program
length

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
```

REM

Purpose

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. Do not use REM commands in frequently-called loops or subroutines.

Syntax

REM

Example

```
>10 REM THIS IS A COMMENT LINE
>20 NEW : REM THIS IS ALSO A COMMENT LINE
```

REN

Purpose

Use the REN command to renumber program lines.

Syntax

REN[new number],[old number],[increment]

Examples

Example	Result
`REN`	Renumbers the entire program. The first new line number is 10. Line numbers increment by 10.
`REN 20`	Renumbers the entire program. The first new line number is 10. Line numbers increment by 20.
`REN 300,50`	Renumbers the entire program. The first new line number is 300. Line numbers increment by 50.
`REN 1000,900,20`	Renumbers the program from line 900 on up. Line number 900 becomes line number 1000. Any following line numbers increment by 20.

ROM

Purpose

Use the ROM command to tell the module interpreter to select the current program out of EEPROM or UVPROM. The current program is displayed during a LIST command and executed when **RUN** is typed.

IMPORTANT Your 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 module selects that program out of EEPROM memory and makes it the current program. If no integer is typed after the ROM command (example: **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 the user wants to run or list. If you attempt to select a program that does not exist (example: 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 the ROM mode is selected. 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 allows you to transfer a program from EEPROM to RAM for editing purposes. You do not get an error message if you attempt to edit a line of RAM program.

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, 71, and 72.You can also use all of the RAM for variable storage if the program is stored in EEPROM. The system control value MTOP always refers to RAM. The system control value LEN refers to the currently selected program in RAM or ROM.

Syntax

ROM[integer]

Example

```
READY
>ROM1
```

RROM

Purpose

Use the RROM command to tell the module interpreter to select the current program out of EEPROM or UVPROM and then execute the program. This command is equivalent to typing **ROM** and then **RUN**.

IMPORTANT Your 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 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 the user wants to run or list. If you attempt to select a program that does not exist (example: 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 allows you to transfer a program from EEPROM to RAM for editing purposes. You do not get an error message if you attempt to edit a line of RAM program.

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, 71, and 72. You can also use all of the RAM for variable storage if the program is stored in EEPROM. The system control value MTOP always refers to RAM. The system control value LEN refers to the currently selected program in RAM or ROM.

Syntax

RROM[integer]

Example

```
READY
>RROM
```

RUN

Purpose

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 command and the GOTO statement are the only ways you can place the module interpreter into Run mode from Command mode. Terminate program execution at any time by pressing **[CTRL-C]** on the console device.

Syntax

RUN

Variations

Some BASIC interpreters allow a line number to follow the RUN command (example: **`RUN 100`**). The 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 Direct mode. See statement GOTO in chapter 7.

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

Purpose

Use the SNGLSTP command to initiate single-step program execution. If the number specified by 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 by typing the RUN command. After each stop, type **`CONT`** 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 EEPROM.

Syntax

SNGLSTP[integer]

Example

```
>1  REM EXAMPLE PROGRAM
>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
>CONT


STOP – LINE 20
READY
>CONT

3

STOP – LINE 30
READY
>SNGLSTP 0

SINGLE STEP DISABLED

READY
>CONT

4
5
PASSED FOR – NEXT LOOP
THIS IS THE END
```

```
READY
>
```

VER

Purpose

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

Syntax

VER

Example

```
>VER
SLC 500 module–Catalog Number 1746-BAS
Firmware release: x.xx
Allen–Bradley Company, Copyright 19xx
All rights reserved

>
```

XFER

Purpose

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

This chapter describes and illustrates CALLs that cause a function to occur within the BASIC or BASIC-T module. These CALLs cannot be executed within the BASIC program *but* are entered at the command line. Table 5.1 lists the corresponding mnemonics.

Table 5.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Battery-backed RAM disable	CALL 73	5-1
Battery-backed RAM enable	CALL 74	5-2
Protected variable storage	CALL 77	5-2
User memory module check and description	CALL 81	5-3
Check the user memory module map.	CALL 82	5-4
Upload the user memory module code to host.	CALL 101	5-4
Print port PRT1 output buffer and pointer.	CALL 103	5-5
Print port PRT1 input buffer and pointer.	CALL 104	5-6
Print the argument stack.	CALL 109	5-7
Print port PRT2 output buffer and pointer.	CALL 110	5-8
Print port PRT2 input buffer and pointer.	CALL 111	5-8

CALL 73 – Battery-Backed RAM Disable

Purpose

Use CALL 73 to disable the battery-backed RAM. When this CALL is executed, the message **`Battery Backup Disabled`** is printed on the host terminal. Disabling battery-backed RAM allows a purging reset. When power to the module is turned OFF, the contents of RAM are destroyed. When power is reapplied, RAM is cleared and battery back-up is re-enabled.

Syntax

CALL 73

Example

```
>CALL 73

Battery Backup Disabled

>REM TURNING POWER OFF, THEN BACK ON
```

CALL 74 – Battery-Backed RAM Enable

Purpose

Use CALL 74 to enable the battery-backed RAM. When this CALL is executed, the message **`Battery Backup Enabled`** is printed on the host terminal. Battery-backed RAM is enabled on module powerup and remains enabled until you execute a CALL 73 or until the battery fails.

Syntax

CALL 74

Example

```
>CALL 74

Battery Backup Enabled
```

CALL 77 – Protected Variable Storage

IMPORTANT Change CALL 77 from Command mode only to ensure proper operation.

Purpose

Use CALL 77 to reserve the top of RAM memory for protected variable storage. Values are saved if BATTERY-BACKUP is invoked. You store values with the ST@ command and retrieve them with the LD@ command. Each variable stored requires 6 bytes of storage space.

You must subtract 6 times the number of variables stored from MTOP reducing available RAM memory. This value is PUSHed onto the stack as the new MTOP address. All appropriate variable pointers are reconsidered. Do this only in Command mode.

IMPORTANT Do not let the ST@ address write over the MTOP address. This could alter the value of a variable or string. The lowest setting MTOP may be set to is 4096 (1000H).

IMPORTANT Call 77 de-allocates all the string memory along with the string contents. Therefore, make sure that you perform this CALL before the execution of the string statement.

Syntax

PUSH [new MTOP address]
CALL 77

Example: (For saving 2 variables)

```
>PRINT MTOP
24575
>PRINT MTOP-12
24563
>PUSH 24563:REM NEW MTOP ADDRESS
>CALL 77

>1   REM EXAMPLE PROGRAM
>10  K = 678*PI
>20  L = 520
>30  PUSH K
>40  ST@ 24575 : REM STORE K IN PROTECTED AREA
>50  PUSH L
>60  ST@ 24569 : REM STORE L IN PROTECTED AREA
>70  REM TO RETRIEVE PROTECTED VARIABLES
>80  LD@ 24575 : REM REMOVE K FROM PROTECTED AREA
>90  POP K
>100 LD@ 24569 : REM REMOVE L FROM PROTECTED AREA
>110 POP L
>120 REM USE LD@ AFTER POWER LOSS AND BATTERY BACK-UP IS USED
```

CALL 81 – User Memory Module Check and Description

Purpose

Use CALL 81 to check the user memory module before burning a program into the memory module. 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 is checksum fails

IMPORTANT CALL 81 cannot detect a defective memory module.

No PUSHes or POPs are needed.

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

Purpose

Use CALL 82 to check the user memory module and display a map of where all the BASIC programs are stored. The programmer can determine by using this CALL where the empty space in the memory module is located and how much space is available. No PUSHes or POPs are needed.

Syntax

CALL 82

Example

```
>CALL 82

 8010H --  805CH --> ROM 1
 805DH --  80A9H --> ROM 2
 80AAH --  80F6H --> ROM 3
 80F7H --  FFFFH --> UNUSED
>
```

CALL 101 – Upload User Memory Module Code to Host

Purpose

Use CALL 101 to upload the code in the user memory module to the host terminal. This CALL requires two PUSHes and no POPs. The first PUSH is the starting address. The second PUSH is the ending address. This CALL converts data within the address range to Intel™ Hex format, then prints the information to the program port. An error message is printed if the addresses are not consistent.

Syntax

PUSH [starting address]
PUSH [ending address]
CALL 101

Example

```
>PUSH 8000 : PUSH 804FH : CALL 101

:108000003107021327CC3313276607005FFF473081
:108010005509000A8B41E034290D1000149C3130C1
:108020002C32302C33302C34300D0A001EA049EA9B
:1080300030A6330D0900289B41E049290D06003286
:1080400097490D0A003CA04AEA4FA6330D090046A5
:00000001F
>
```

CALL 103 – Print PRT1 Output Buffer and Pointer

Purpose

Use CALL 103 to print the complete output buffer with address, front pointer, and number of characters in the buffer to the program port screen. No PUSHes or POPs are needed.

Use this information as a troubleshooting aid. It does not affect the contents of the buffer.

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 Input Buffer and Pointer

Purpose

Use CALL 104 to print the complete input buffer with address, front pointer, and number of characters in the buffer to the program port screen. No PUSHes or POPs are needed.

Use this information as a troubleshooting aid. It does not affect the contents of the buffer.

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 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 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 109 – Print Argument Stack

Purpose

Use CALL 109 to print the top 9 values on the argument stack to the console. No PUSHes or POPs are needed. Use this information as a troubleshooting aid. It does not affect the contents of the argument stack or pointer to the stack.

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 41H 67H 50H 00H 00H 7EH
 1E1H 83H 75H 00H 00H 00H 7CH
 1E7H 13H 04H 00H 00H 00H 7CH
 1EDH 32H 84H 70H 00H 00H 85H
 1F3H 32H 76H 80H 00H 00H 85H
 1F9H 00H 00H 00H 00H 00H 00H

Argument stack pointer is: 01FEH
```

CALL 110 – Print PRT2 Output Buffer Pointer

Purpose

Use CALL 110 to print the complete output buffer with addresses, front pointer and the number of characters in the buffer to the console device. No PUSHes or POPs are needed.

Use this information as a troubleshooting aid. It does not affect the contents of the buffer.

Syntax

CALL 110

Example

```
>CALL 110
                          PRT2 Output Queue

6F00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F10H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F20H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F30H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F40H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F50H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F60H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F70H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F80H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6F90H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FA0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FB0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FC0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FD0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FE0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6FF0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H

Output queue front pointer is: 6F00H
```

CALL 111 – Print PRT2 Input Buffer Pointer

Purpose

Use CALL 111 to print the complete input buffer with addresses, front pointer and the number of characters in the buffer to the console device. No PUSHes or POPs are needed.

Use this information as a troubleshooting aid. It does not affect the contents of the buffer.

Syntax

CALL 111

Example

```
>CALL 111
                          PRT2 Input Queue

6E00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E10H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E20H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E30H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E40H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E50H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E60H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E70H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E80H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6E90H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EA0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EB0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EC0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6ED0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EE0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
6EF0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H

Input queue front pointer is: 6E00H
```

Assignment Functions

This chapter describes and illustrates commands that assign storage, reset data storage, and values to variables within the BASIC program or from the command line. Table 6.1 lists the corresponding mnemonics.

Table 6.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Clear variables, interrupts & strings.	CLEAR	6-1
Clear interrupts.	CLEARI	6-3
Clear all stacks.	CLEARS	6-3
Data read by Read statement.	DATA	6-4
Allocate memory for array variables.	DIM	6-4
Assign a variable or a string a value (LET is optional).	LET	6-5
RESTORE read pointer.	RESTORE	6-7

CLEAR

Purpose

Use the CLEAR statement to set all variables equal to 0 and reset all BASIC evoked interrupts and stacks. This means that after the CLEAR statement is executed, an ONTIME statement must be executed before the module acknowledges the internal timer interrupts. ERROR trapping with the ONERR statement also does not occur until an ONERR [ln num] statement is executed.

The CLEAR statement does not affect the free running clock that is enabled by the CLOCK1 statement. CLOCK0 is the only module statement that can disable the free running clock.

CLEAR also does not reset the memory that has been allocated for strings, so it is not necessary to enter the STRING [exp], [expr] statement to re-allocate memory for strings after the CLEAR statement is executed. In general, CLEAR is used to erase all variables.

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=O 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

Purpose

Use the CLEARI statement to clear all of the BASIC evoked interrupts. The ONTIME interrupt is disabled after the CLEARI statement is executed.

CLEARI does not affect the free running clock enabled by the CLOCK1 statement. CLOCK0 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.

IMPORTANT When the CLEARI statement is LISTed it appears as CLEARI.

Syntax

CLEARI

Example

```
READY
>CLEARI
```

CLEARS

Purpose

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.

IMPORTANT When the CLEARS statement is LISTed it appears as CLEARS.

Syntax

CLEARS

Example

```
READY
>CLEARS
```

DATA

Purpose

Use the DATA statement to specify the expressions that you can retrieve with a READ statement. If multiple expressions per line are used, 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 considered chained and appear as one large DATA statement. If at anytime all the data is read and another READ statement is executed, the program terminates and the message **`ERROR: NO DATA - IN LINE XX`** prints to the console device. The module returns to Command mode.

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=O TO 3
70 PRINT A(J)
80 NEXT J

READY
>RUN

 10
 65
 67
 35.627
```

IMPORTANT You cannot use the CHR operator in a DATA statement.

DIM

Purpose

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

Once a variable is dimensioned in a program it may not be re-dimensioned. An attempt to re-dimension an array causes an array size error that causes the module to enter the Command mode.

If an array variable is used that was not dimensioned by a DIM statement, BASIC assigns a default value of 10 to the array size. All arrays are set equal to zero when the RUN command, NEW command or the CLEAR statement is executed.

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

Syntax

DIM

Examples

More than one variable can be dimensioned by a single DIM statement.

```
>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
>
```

LET

Purpose

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

Syntax

LET [var] = [expr]

Examples

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

>1  REM EXAMPLE PROGRAM
>10 LET A = A +1
```

NOTE The - sign used in the LET statement is not an equality operator. It is a replacement operator. The statement should be read A is replaced by A plus one. The word LET is always optional (example: **`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 or else a memory allocation error occurs that causes the module to enter the Command mode.

RESTORE

Purpose

Use the RESTORE statement to reset the internal read pointer to the beginning of the data so that it may be read again.

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
```

Control Functions

This chapter describes and illustrates commands executed within the BASIC program or from the command line to control the internal clock or the flow of the BASIC program. Table 7.1 lists the corresponding mnemonics.

Table 7.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Disable the real time clock.	CLOCK0	7-2
Enable the real time clock.	CLOCK1	7-1
Set up a conditional do-loop.	DO-UNTIL	7-4
Set up a conditional do-loop.	DO-WHILE	7-3
Terminate a program execution.	END	7-5
Set up a for-next loop.	FOR-TO-(STEP)-NEXT	7-6
Go to the program line number.	GOTO	7-7
Conditional test	IF-THEN-ELSE	7-8
Test a for-next loop condition.	NEXT	7-9
Conditional GOTO	ON-GOTO	7-11

CLOCK1

Purpose

Use the CLOCK1 statement to enable the free running clock resident on the BASIC or BASIC-T module. The special function operator TIME is incremented once every 5 milliseconds after the CLOCK1 statement is executed. The CLOCK1 statement 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 zero. The interrupts associated with the CLOCK1 statement 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 ON-TIME user interrupt handling execution.

Syntax

CLOCK1

Example

```
>NEW

>1   REM EXAMPLE PROGRAM
>10  TIME = 0
>15  DBY(71) = 0
>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
```

CLOCK0

Purpose

Use the CLOCK0 (zero) statement to disable or turn off the free running clock resident on the BASIC module. After CLOCK0 is executed, the special function operator TIME no longer increments. CLOCK0 is the only module statement that can disable the free running clock. CLEAR and CLEARI do *not* disable the free running clock, only its associated ONTIME interrupt.

IMPORTANT CLOCK1 and CLOCK0 are independent of the clock/calendar.

Syntax

CLOCK0

Example

```
READY
>CLOCK0
```

DO-WHILE

Purpose

Use the DO-WHILE statement to set up loop control within a module program. The operation of this statement is similar to the DO-UNTIL [rel expr]. 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 (example: DO-WHILE, DO-UNTIL, FOR-NEXT and BASIC subroutines). The control stack is 157 bytes long. DO-WHILE and DO-UNTIL loops and GOSUB commands use 3 bytes of the control stack. FOR-NEXT loops use 17 bytes.

IMPORTANT Excessive nesting exceeds the limits of the control stack, generating an error, and causing the module to enter Command mode.

Syntax

DO-WHILE [rel expr]

Examples

Simple DO-WHILE

```
>NEW

>1  REM EXAMPLE PROGRAM
>10 DO
>20 A = A + 1
>30 PRINT A
>40 WHILE A < 4
>50 PRINT “DONE”
>60 END

READY
>RUN

 1
 2
 3
 4
DONE

READY
>
```

Nested DO-WHILE

```
>NEW
```

```
>1  REM EXAMPLE PROGRAM
>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

READY
>
```

DO-UNTIL

Purpose

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 (example: DO-WHILE, DO-UNTIL, FOR-NEXT and BASIC subroutines). The control stack is 157 bytes long. DO-WHILE and DO-UNTIL loops and GOSUB commands use 3 bytes of the control stack. FOR-NEXT loops use 17 bytes.

IMPORTANT Excessive nesting exceeds the limits of the control stack, generating an error, and causing the module to enter Command mode.

Syntax

DO-UNTIL [rel expr]

Examples

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
```

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
```

END

Purpose

Use the END statement to terminate program execution. CONT does not operate if the END statement is used 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

End Statement Termination

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

READY
>RUN

 1  2  3  4
READY
>
```

FOR-TO-(STEP)-NEXT

Purpose

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

The control stack (C-stack) stores all information associated with loop control (example: DO-WHILE, DO-UNTIL, FOR-NEXT and BASIC subroutines). The control stack is 157 bytes long. DO-WHILE and DO-UNTIL loops and GOSUB commands use 3 bytes of the control stack. FOR-NEXT loops use 17 bytes.

IMPORTANT Excessive nesting exceeds the limits of the control stack, generating an error, and causing the module to enter Command mode.

Syntax

FOR [expr] TO [expr] STEP [expr]
.
.
.
NEXT [expr]

Examples

```
>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
```

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

READY
>RUN

 1  2  3  4
```

In the first example, 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.

If the STEP statement and the value D are omitted, the increment value defaults to 1, therefore; STEP is an optional statement. The NEXT statement returns the loop to the beginning of the loop and adds the value of D to the current index value. The current index value is then compared to the value of C, 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 module. The NEXT statement is always followed by the appropriate variable. You may nest FOR-NEXT loops up to 9 times.

```
>1  REM EXAMPLE PROGRAM
>10 FOR I=1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END
>RUN
>1 2 3 4


READY
```

```
>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

READY
```

GOTO

Purpose

Use the GOTO statement to cause BASIC to transfer control to the line number ([ln num]) specified.

Syntax

GOTO [ln num]

Example

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

If line 100 exists, this statement causes execution of the program to resume at line 100. If line number 100 does not exist, the message **ERROR: INVALID LINE NUMBER** is printed to the console device and the module enters the Command mode.

Unlike the RUN command, the GOTO statement, if executed in the Command mode, does not clear the variable storage space or interrupts. However, if the GOTO statement is executed in the Command mode after a line is edited, the module clears the variable storage space and all BASIC evoked interrupts.

IF-THEN-ELSE

Purpose

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

Syntax

IF [rel expr] THEN valid statement ELSE valid statement

Examples

Example 1

```
>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. If you want to transfer control to different line numbers using the IF statement, you may omit the GOTO statement. The following 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 module statement as shown below:

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

Example 2

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

In these examples, if A does not equal 10, then both PRINT A and GOTO 150 are executed. If A equals 10, then control passes to 10.

Example 3

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
```

In this example, if A equals 10 then control passes to line number 40. If A does not equal 10, line number 30 is executed.

NEXT

Purpose

Use the NEXT statement to return the FOR-TO-(STEP)-NEXT loop 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
```

ON-GOTO

Purpose

Use the ON-GOTO statement to transfer control to the line(s) specified by the GOTO statement when the value of the expression following the ON statement is encountered in the BASIC 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. All comments that apply to GOTO apply to the ON statement. If Q is less than zero, an **ERROR: BAD ARGUMENT** message is generated and the BASIC module enters Command mode. If Q 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 module program.

Execution Control and Interrupt Support Functions

This chapter describes and illustrates commands that control data flow and program transfer between ROM and RAM within the BASIC program or from the command line. Table 8.1 lists the corresponding mnemonics.

Table 8.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Enable the interrupt capability when a DF1 packet is received.	CALL 16	8-2
Disable the DF1 packet interrupt capability.	CALL 17	8-3
Enable the SLC processor interrupt capability.	CALL 20	8-3
Disable the SLC processor interrupt capability.	CALL 21	8-4
Generate an interrupt to the SLC processor.	CALL 26	8-4
Initiate transactions defined by CALLs 27, 28, 122, and 123.	CALL 38	8-5
ROM to RAM program transfer	CALL 70	8-8
ROM/RAM to ROM program transfer	CALL 71	8-9
RAM/ROM return	CALL 72	8-9
Execute a subroutine.	GOSUB	8-11
Go to line number when an error is detected.	ONERR	8-12
Conditional GOSUB	ON-GOSUB	8-14
Generate an interrupt when TIME is equal to or greater than ONTIME argument-line number.	ONTIME	8-14
POP argument stack to variables.	POP	8-17
PUSH expressions on argument stack.	PUSH	8-15
Return from interrupt.	RETI	8-18
RETURN from subroutine.	RETURN	8-18
Break program execution.	STOP	8-20

CALL 16 – Enable DF1 Packet Interrupt

Purpose

Use CALL 16 to enable the DF1 packet interrupt capability. One argument is PUSHed and no arguments are POPped. The input argument is the BASIC line number of the beginning of the interrupt routine that the program should jump to, when a valid DF1 packet is received in the port PRT2 buffer. You should process the packet within the interrupt routine. A RETI executed within the routine returns you to the point in the program before the interrupt occurred. This command has no effect if the DF1 protocol is not enabled (CALL 108). Also, jumper JW4 must be in a position enabling DF1 for port PRT2.

Once this CALL is enabled, port PRT2 is checked by the processor at the end of each line of BASIC code for a DF1 message received.

If the DF1 packet arrives due to CALL 122 or CALL 123 when CALL 16 is enabled, you will receive the DF1 packet interrupt but the DF1 packet will have been removed from the input buffer.

Interrupts are disabled when the module is in Command mode. CALL 16 disabled is the default of the module when entering Run mode. CALL 16 must be re-executed every time Run mode is entered.

Syntax

PUSH [BASIC line number]
CALL 16

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PACKET INTERRUPT
>20  PUSH 800: REM LINE NUMBER OF START OF DF1 INTERRUPT ROUTINE
>30  CALL 16
>800 (BEGINNING OF INTERRUPT ROUTINE)
 :   (PROCESS THE PACKET)
>850 RETI
```

CALL 17 – Disable DF1 Packet Interrupt

Purpose

Use CALL 17 to disable the DF1 packet interrupt capability enabled with CALL 16. This routine has no input or output arguments.

Syntax

CALL 17

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM DISABLE DF1 PACKET INTERRUPT ENABLED WITH CALL 16
>20 CALL 17
```

CALL 20 – Enable Processor Interrupt

Purpose

Use CALL 20 to allow the processor to interrupt the module. One argument is PUSHed and no arguments are POPped. The PUSH is the BASIC line number of the beginning of the interrupt routine that the program should jump to, when word 0, bit 15 in the CPU output image table, toggles from a low to a high value. The module detects this transition automatically and jumps to an interrupt routine. A RETI executed within the interrupt routine returns you to the point in the module program before the interrupt occurred.

The module monitors CPU output file word 0, bit 15 at the end of every BASIC line and generates the interrupt if the bit goes high. The CPU must hold the interrupt request bit low for at least 10 milliseconds prior to requesting interrupt service. The bit must be held high for at least one scan so the bit can be detected by the module.

If another interrupt is detected before the previous one is fully serviced, the new interrupt is marked pending. Only one interrupt is pending at a time.

Interrupts are disabled when the module is in Command mode. CALL 20 disabled is the default of the module when entering Run mode. CALL 20 must be re-executed every time Run mode is entered.

Syntax

PUSH [BASIC line number]
CALL 20

Example

```
>1    REM EXAMPLE PROGRAM
>10   REM ENABLE PROCESSOR INTERRUPTS
>20   PUSH 1000 : REM LINE NUMBER OF START OF PROCESSOR
      INTERRUPT ROUTINE
>30   CALL 20
>1000 (BEGINNING OF THE PROCESSOR INTERRUPT ROUTINE)
 :
>1050 RETI
```

CALL 21 – Disable Processor Interrupt

Purpose

Use CALL 21 to disable the processor interrupt capability enabled with CALL 20. This routine has no input or output arguments.

Syntax

CALL 21

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM DISABLE PROCESSOR INTERRUPTS ENABLED WITH CALL 20
>20 CALL 21
```

CALL 26 – Module Interrupt

Purpose

Use CALL 26 to generate an interrupt to the SLC 5/02™ and above processors. No arguments are PUSHed and one argument is POPped. The POP shows the status of the SLC processor. When this CALL is executed, an I/O event interrupt is issued by the module to interrupt the normal processor operating cycle in order to scan a specified subroutine. This interrupt causes the SLC processor to execute the interrupt subroutine file configured in the module slot configuration (ISR numbered file). The module remains in this CALL routine until an interrupt acknowledge is received from the processor. CALL 26 must be executed in the BASIC program each time the SLC processor is to be interrupted.

This CALL has no effect if the SLC processor is not in the Run mode.

After the module issues the CALL, it may take up to 5 milliseconds for the execution of interrupt to occur.

The POP shows the status of the SLC processor:

- 0 - SLC processor acknowledges the interrupt but may not have executed the interrupt routine yet
- 1 - SLC processor aborted the interrupt
- 2 - SLC processor is not in Run mode
- 3 - SLC 500 fixed and SLC 5/01™ processor cannot support interrupts

Syntax

CALL 26

POP[SLC processor status]

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE MODULE INTERRUPT TO THE SLC PROCESSOR TO EXECUTE
THE ISR FILE
>20 CALL 26
>30 POP S : REM SLC PROCESSOR STATUS
```

CALL 38 – Expanded ONERR Restart

Purpose

Use CALL 38 to expand the type of errors trapped and handled by the ONERR function. One argument is PUSHed and no arguments are POPped. The ONERR restart allows the module to jump to an error handling routine when the arithmetic overflow, divide by zero, and bad argument errors are encountered. All other errors cause the module to enter the Command mode and stop the execution of the program. When CALL 38 is enabled, all errors other than hardware-specific failures (watchdog, RAM failure, etc.), cause the program to enter the routine defined in the ONERR function, instead of returning to the Command mode. This routine may be used to reset the error and resume normal program operation. If any error occurs that causes a restart, stacks are cleared and variables and ports are not re-initialized. This CALL has no effect until the ONERR command is executed within the program.

The PUSH determines if this expanded ONERR function is enabled or disabled as shown below:

- 0 - Disable the expanded ONERR restart
- 1 (or any other number) - Enable the expanded restart

This CALL is reset when the module returns to the Command mode. CALL 38 must be re-executed every time Run mode is entered.

If you perform an XBY in the error routine, this is a list of the status codes you might receive.

Table 8.2 Status Codes

Status Code	Description
01	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	DH485 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
17	no DATA available for READ
18	DF1 cannot be enabled (JW4 in wrong position)
19	• illegal user 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

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE EXPANDED ONERR FUNCTION
>20  ONERR 160
>30  PUSH 1
>40  CALL 38
>50  CALL 53: REM GET DATA FROM OUTPUT IMAGE
>60  PUSH 201: REM ADDRESS OF SECOND WORD IN BUFFER
>70  CALL 14: REM GET DATA FROM INPUT BUFFER
>80  POP X: REM VALUE FROM INPUT BUFFER
>90  A=(X*2.499733)-8191.625
>100 PUSH A: REM RESULT OF ABOVE CALCULATION
>110 PUSH 201: REM WORD NUMBER OF BASIC OUTPUT BUFFER
>120 CALL 24: REM BASIC FLOATING POINT TO 16-BIT SIGNED
INTEGER
>130 CALL 54: REM OUTPUT BUFFER TO SLC INPUT FILE
>140 POP Y: REM SLC PROCESSOR STATUS
>150 IF (Y<>0) THEN PRINT “PROCESSOR NOT IN RUN MODE”
>160 GOTO 50
>170 PRINT “ERROR CODE WAS”,XBY(257) : REM BEGINNING OF ONERR
ROUTINE
>175 PRINT “AT LINE “, (256*XBY (69FDH) + XBY(69FEH))
>180 GOTO 50
>190 END
```

The error in the example above is a missing POP for CALL 53. The missing POP will cause an A-Stack error and would normally put the processor in Command mode. When this occurs, print the error code and resume running the program.

CALL 70 – ROM to RAM Program Transfer

Purpose

Use CALL 70 to shift program execution from a running ROM program to the beginning of the RAM program. No arguments are PUSHed or POPped.

IMPORTANT The first line of the RAM program is not executed. We recommend that you make it a remark.

Syntax

CALL 70

Example

```
READY
>LIST
1  REM EXAMPLE PROGRAM
10 REM SAMPLE ROM PROGRAM FOR CALL 70
20 PRINT “NOW EXECUTING ROM 5”
30 CALL 70 : REM GO EXECUTE RAM
40 END

READY
>RUN

NOW EXECUTING ROM 5
NOW EXECUTING RAM

READY
>LIST
1  REM EXAMPLE PROGRAM
10 REM SAMPLE RAM PROGRAM FOR CALL 70
20 PRINT “NOW EXECUTING RAM”
30 END

READY
```

CALL 71 – ROM/RAM to ROM Program Transfer

Purpose

Use CALL 71 to transfer from a running ROM or RAM program to the beginning of any available ROM program. One argument is PUSHed (which ROM program). None are POPped. An invalid program error displays and you enter the Command mode if the ROM number does not exist.

IMPORTANT The first line of the ROM program is not executed. We recommend that you make it a remark.

Syntax

PUSH [ROM program number]
CALL 71

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM THIS ROUTINE WILL CALL AND EXECUTE 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
```

You are 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

Purpose

Use CALL 72 to return to the ROM or RAM routine that called this ROM or RAM routine. Execution begins on the line following the line that CALLed the routine. No arguments are PUSHed or POPped. This routine works one layer deep. Program control reverts to the line following the CALL in the previous program.

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.

Syntax

CALL 72

Example

Program in ROM 1

```
>1  REM EXAMPLE PROGRAM
>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
```

Program in ROM 3

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “NOW EXECUTING ROM 3”
>20 CALL 72
>30 END
```

With ROM 1 selected:

```
>RUN

NOW EXECUTING ROM 1
NOW EXECUTING ROM 3
EXECUTING ROM 1 AGAIN

READY
>
```

GOSUB

Purpose

Use the GOSUB statement to cause the module to transfer control of the program to the line number [ln num] following the GOSUB statement. In addition, the GOSUB statement saves the location of the statement following GOSUB on the control stack so that you can perform a RETURN statement to return control to the statement following the most recently executed GOSUB statement. You may nest the GOSUB statement up to 9 times.

The control stack (C-stack) stores all information associated with loop control (example: DO-WHILE, DO-UNTIL, FOR-NEXT and BASIC subroutines). The control stack is 157 bytes long. DO-WHILE and DO-UNTIL loops and GOSUB commands use 3 bytes of the control stack. FOR-NEXT loops use 17 bytes.

IMPORTANT Excessive nesting exceeds the limits of the control stack, generating an error, and causing the module to enter Command mode.

Syntax

GOSUB [ln num]

Examples

Simple Subroutine

```
READY
>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

READY
>NEW
```

Nested Subroutine

```
>1   REM EXAMPLE PROGRAM
>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

READY
>
```

ONERR

Purpose

Use the ONERR statement to handle arithmetic errors, if they occur, during program execution. Only arithmetic overflow, arithmetic underflow, divide by zero, and bad argument errors are trapped by the ONERR statement. All other errors are not trapped and cause the module to enter the Command mode. If an arithmetic error occurs after the ONERR statement is executed, 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 8-5.

After the ONERR statement is executed, 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 = 20-ARITH. OVERFLOW or ARITH.UNDERFLOW
- ERROR CODE = 40-BAD ARGUMENT

You can examine this location by using an XBY(257) statement.

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 END

READY
>RUN

.2
.25
.33333333
.5
 1
ERROR CODE WAS 10

READY
>
```

A GOTO statement can replace the END statement in this example to provide a method of user programmed error recovery.

ON-GOSUB

Purpose

Use the ON-GOSUB statement to transfer control to the line(s) specified by the GOSUB statement when the value of the expression following the ON statement is encountered in the BASIC program.

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. All comments that apply to GOSUB apply to the ON statement. If Q is less than zero an **`ERROR: BAD ARGUMENT`** message is generated. If Q 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 module program.

ONTIME

Purpose

Use the ONTIME [expr], [ln num] statement to compensate for the incompatibility between the timer/counters on the microprocessor and the module. Your 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, 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 to the line number [ln num] following the expression [expr] in the ONTIME statement.

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 for ONTIME to operate. If CLOCK1 is not executed the special function operator, TIME, does not increment.

Syntax

ONTIME [expr], [ln num]

Example

```
>1   REM EXAMPLE PROGRAM
>10  TIME = 0
>15  DBY(71) = 0
>20  CLOCK1
>30  ONTIME 2,100
>40  DO
>50  WHILE TIME < 10
>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
TIMER INTERRUPT AT - 8.01 SECONDS
TIMER INTERRUPT AT - 10.01 SECONDS
```

In the example above, the time printed out is .01 seconds later than the time that was supposed to be printed. This is caused by the terminal used in the example operating at 19200 baud which causes a .01 second delay in printing.

To execute the ONTIME interrupt at a fraction of a second use DBY(71) - X where X - 0 to 200. Each count represents a 5 millisecond time interval.

PUSH

Purpose

Use the PUSH statement to place the arithmetic expression or expressions in the 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.

The PUSH and POP statements provide a simple means of passing parameters to CALL routines. 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]). Each expression must be followed by a comma. The last value PUSHed onto the argument stack is the last expression [expr] 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

>1  REM EXAMPLE PROGRAM
>10 PUSH 0
>20 CALL 14
>30 POP W
>40 PRINT W
>50 END

READY
>RUN

 0

READY
>
```

POP

Purpose

Use the POP statement to remove values from the 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 (example: incremented by 6). You can place values in the stack using the PUSH statement.

IMPORTANT If a POP statement executes and no number is in the argument stack, an A-Stack error occurs and the 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]). Each expression must be followed by a comma.

Syntax

POP [var], [var],.....[var]

Example

See the PUSH statement.

You can use the PUSH and POP statements to minimize GLOBAL variable problems. These are caused by the main program and all main program subroutines using the same variable names (example: GLOBAL VARIABLES). 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 is executed.

If you reserve some variable names just for subroutines (S1, S2) and pass variables on the stack as shown in the previous example, you can avoid any GLOBAL variable problems in the module.

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 in which large amounts of data must be PUSHed or POPped, as shown below.

```
>1  REM EXAMPLE PROGRAM
>40 FOR I=1 TO 64
>50 PUSH I
>60 CALL 10
>70 POP A(I)
>80 NEXT I
```

RETI

Purpose

Use the RETI statement to exit from an interrupt (ONTIME, CALL 16, or CALL 20) that is processed in a module program. The RETI statement functions the same as the RETURN statement except that it also clears a software interrupt flag so interrupts can again be acknowledged. If you fail to 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

Purpose

Use the RETURN statement to return control to the statement following the most recently executed GOSUB STATEMENT. Use one return for each GOSUB to avoid overflowing the control stack. This means that a subroutine called by the GOSUB statement can call another subroutine with another GOSUB statement.

Syntax

RETURN

Examples

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

READY
>
```

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

READY
>
```

STOP

Purpose

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 CONTinue command. The purpose of the STOP statement is to allow for easy program debugging.

Syntax

STOP

Example

```
1
STOP - IN LINE 40

>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
>
```

NOTE 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.

Math and Backplane Conversion Functions

This chapter describes and illustrates commands that convert numbers between integer and BASIC floating-point. This chapter also describes and illustrates commands that transfer data from the BASIC or BASIC-T module output buffer to the processor input image or transfers data from the output image to the module input buffer. These commands can be used within the BASIC program or from the command line. Table 9.1 lists the corresponding mnemonics.

Table 9.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Convert 16-bit signed integer to BASIC floating-point.	CALL 14	9-1
Convert 16-bit unsigned integer to BASIC floating-point.	CALL 15	9-2
Convert BASIC floating-point to 16-bit signed integer.	CALL 24	9-2
Convert BASIC floating-point to 16-bit binary.	CALL 25	9-3
Convert BASIC floating-point to SLC floating-point	CALL 88	9-4
Convert SLC floating-point to BASIC floating-point	CALL 89	9-5

CALL 14 – 16-Bit Signed Integer to BASIC Floating-Point

Purpose

Use CALL 14 to convert an SLC 500 controller 16-bit signed integer number to a BASIC floating-point number. The input argument is the address number (0 to 207) of the word in the module input buffer to be converted. The output argument is the converted value.

Syntax

PUSH [word number of module input buffer]
CALL 14
POP [converted value]

Example

```
>1  REM EXAMPLE PROGRAM
>20 PUSH 0 : REM CONVERT 1ST WORD 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
>RUN

 0

READY
>
```

CALL 15 – 16-Bit Unsigned Integer to BASIC Floating-Point

Purpose

Use CALL 15 to convert an SLC 500 controller 16-bit unsigned integer number to a module floating-point number. The input argument is the address number (0 to 207) of the word in the module input buffer to be converted. The output argument is the converted value.

Syntax

PUSH [word number of module input buffer]
CALL 15
POP [converted value]

Example

```
>1  REM EXAMPLE PROGRAM
>50 PUSH 9 : REM CONVERT 10TH WORD OF BASIC INPUT BUFFER
>60 CALL 15 : REM DO 16-BIT UNSIGNED INTEGER TO
    F.P. CONVERSION
>70 POP W : REM GET CONVERTED VALUE
>80 PRINT W
>RUN

 0

READY
>
```

CALL 24 – BASIC Floating-Point to 16-Bit Signed Integer

Purpose

Use CALL 24 to convert a module floating–point number to a signed 16-bit integer and place the result in the module output buffer. The first value PUSHed is the data variable. The second value PUSHed is the address number (0 to 207) of the word in the module output buffer.

IMPORTANT If an attempt is made to write to word 200 of the BASIC output buffer, an error message is displayed and the module returns to Command mode. The bits of word 200 are defined.

The fractional part of the module floating-point value is truncated. If the module floating-point value is less than –32768, the value placed in the module output buffer is –32768. If the module floating-point value is greater than +32767, the value placed in the module output buffer is +32767. The programmer is responsible for checking the range of the number before conversion.

Syntax

PUSH [value to be converted]
PUSH [word number of module output buffer]
CALL 24

Example

```
>1  REM EXAMPLE PROGRAM
>10 W = 17
>40 PUSH W : REM THE VALUE TO BE CONVERTED
>50 PUSH 0 : REM 1ST WORD OF BASIC OUTPUT BUFFER
>60 CALL 24 : REM DO THE F.P. TO 16-BIT SIGNED CONVERSION

READY
>
```

CALL 25 – BASIC Floating-Point to 16-Bit Binary

Purpose

Use CALL 25 to convert a module floating-point value between 0 and 65535 to its 16-bit binary representation. The resulting value is then stored in the module output buffer. The first value PUSHed is the data to be converted. The second value PUSHed is the address number (0 to 207) of the word in the module output buffer.

IMPORTANT If an attempt is made to write to word 200 of the BASIC output buffer, an error message is displayed and the module returns to Command mode. The bits of word 200 are defined.

The fractional part of the module floating-point value is truncated. If the module floating-point value is less than 0, then the value placed in the module output buffer is 0. If the value is greater than +65535, then the value placed in the module output buffer is +65535. The programmer is responsible for checking the range of the number before conversion.

Syntax

PUSH [value to be converted]
PUSH [word number of module output buffer]
CALL 25

Example

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 9E+1 : REM THE VALUE TO BE CONVERTED
>50 PUSH 0 : REM 1ST WORD OF BASIC OUTPUT BUFFER
>60 CALL 25 : REM DO F.P. TO 16-BIT BINARY CONVERSION
>

READY
>
```

CALL 88: BASIC Floating-Point to SLC Floating-Point

Purpose

This CALL may only be used with a SLC™ 5/03, 5/04 or 5/05 processor. These are the only SLC processors that support the floating-point data type. In addition, the module must be configured for SLC 5/02 mode (Class 4), so it cannot be used in a remote I/O chassis with a 1747-ASB. This CALL may be used in a remote ControlNet chassis with a 1747-ACN(R)15.

Use this CALL to convert BASIC floating-point to SLC floating-point in a two-word format and place the converted value in the module's output buffer. See also CALL 89.

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

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

The SLC floating-point number is a 7-digit binary floating-point number (IEEE Float 32-bit value). The range of the SLC floating-point number:

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

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

SLC floating-point numbers are stored in 2 16-bit words.

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

Syntax

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 module's output buffer. The output buffer addresses are 100-163 for the M1 file and 200-207 for the Input Image.

PUSH [number to convert]

PUSH [output buffer to receive converted value]

CALL 88

Example

```
>50  PUSH F: REM Floating-point value to convert

>60  PUSH 100 : REM Words 100 and 101 of the BASIC output buffer
associated with the M1 file

>70  CALL 88 :

>80  PUSH 2 : REM Number of words to transfer from BASIC output
buffer to M1 file

>90  CALL 57 : REM Transfer data from BASIC output buffer to M1
file

>100 POP S : REM status for CALL 57
```

CALL 89: SLC Floating-Point to BASIC Floating-Point

Purpose

This CALL may only be used with a SLC™ 5/03, 5/04 or 5/05 processor. These are the only SLC processors that support the floating-point data type. In addition, the module must be configured for SLC 5/02 mode (Class 4), so it cannot be used in a remote I/O chassis with a 1747-ASB. This CALL may be used in a remote ControlNet chassis with a 1747-ACN(R)15.

Use this call to convert SLC floating-point to BASIC floating-point. See also CALL 88.

The SLC floating-point number is a 7-digit binary floating-point number (IEEE Float 32-bit value). The range of the SLC floating-point number:

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

The module floating-point number is an 8-digit BCD floating-point number. The range of the module floating-point number is $\pm 1E^{-127}$ to $\pm .99999999E^{+127}$

SLC floating-point numbers are stored in two 16-bit words.

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

Syntax

This routine has one input and one output argument. The input argument is the address of the module's input buffer containing the value to be converted. Addresses 100-163 are from the M0 file and 200-207 are from the SLC Output Image. The output argument is the converted value.

PUSH [input buffer of value to be converted]

CALL 89

POP converted value

Example

```
>50   PUSH 2: REM Transfer 2 words (1 floating-point value) from
M0 file to BASIC input buffer for conversion

>60  CALL 56: REM Transfers data from M0 file to BASIC input
buffer

>70  POP 5 : REM Status for CALL 56

>80  PUSH 100 : REM module input buffer address

>90  CALL 89

>100 POP C : REM The variable C will contain the converted value
```

Clock/Calendar Functions

This chapter describes and illustrates commands that set and display the real time clock/calendar within the BASIC program or from the command line. Table 10.1 lists the corresponding mnemonics.

Table 10.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Set the clock/calendar time (hour, minute, second).	CALL 40	10-1
Set the clock/calendar date (day, month, year).	CALL 41	10-2
Set the clock/calendar - day of week.	CALL 42	10-3
Retrieve a date/time string.	CALL 43	10-4
Retrieve the date numeric (day, month, year).	CALL 44	10-4
Retrieve a time string.	CALL 45	10-5
Retrieve a time numeric.	CALL 46	10-6
Retrieve the day of week string.	CALL 47	10-6
Retrieve the day of week numeric.	CALL 48	10-7
Retrieve a date string.	CALL 52	10-7

CALL 40 – Set Clock/ Calendar Time

Purpose

Use CALL 40 to set the following clock/calendar time functions:

- H - hours (0 to 23; only a 24-hour clock is available)
- M - minutes (0 to 59)
- S - seconds (0 to 59)

Syntax

PUSH [hours]
PUSH [minutes]
PUSH [seconds]
CALL 40

Example

Program the wall clock for 1:35 P.M. (programmed as 13:35; only a 24-hour clock is available)

```
>1  REM EXAMPLE PROGRAM
>10 H = 13 : M = 35 : S = 00
>20 REM HOURS = 13, MINUTES = 35, SECONDS = 00
>30 PUSH H,M,S
>40 CALL 40

READY
>RUN

READY
>
```

CALL 41 – Set Clock/ Calendar Date

Purpose

Use CALL 41 to set the following clock/calendar functions:

- D - day
- M - month
- Y - year

Three values are PUSHed and none are POPped.

Syntax

PUSH [day]
PUSH [month]
PUSH [year]
CALL 41

Example

Program the clock/calendar for the 16th day of June 1991.

```
>1   REM EXAMPLE PROGRAM
>5   STRING 100,20
>10  H=13 : M=35 : S=00
>20  REM HOURS = 13, MINUTES = 35, SECONDS = 00
>30  PUSH H,M,S
>40  CALL 40
>60  D=16 : MO=6 : Y=91
>70  PUSH D,MO,Y
>80  CALL 41
>90  PUSH 3
>100 CALL 42
>120 PUSH 0
>130 CALL 43
>140 PRINT $(0)

READY
>RUN

16-JUN-91 13:35:00
```

CALL 42 – Set Day of Week

Purpose

Use CALL 42 to set the day of the week. Sunday is day 1. Saturday is day 7.

Syntax

PUSH [day of week]
CALL 42

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 3: CALL 42:REM DAY IS TUESDAY.
```

CALL 43 – Retrieve Date/ Time String

Purpose

Use CALL 43 to return the current date and time as a string. PUSH the number of the string to receive the date/time (dd-mmm-yy HH:MM:SS). You must 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 [string number]
CALL 43

Example

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1: CALL 43: REM PUT DATE/TIME IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

16-JUN-91 13:35:00

READY
>
```

CALL 44 – Retrieve Date Numeric

Purpose

Use CALL 44 to return the current date on the argument stack as three numbers. There is no input argument to this routine and three variables are returned. The date is POPped in day, month and year order.

Syntax

CALL 44
POP [day]
POP [month]
POP [year]

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM DATE RETRIEVE - NUMERIC EXAMPLE
>20 CALL 44 : REM INVOKE THE UTILITY ROUTINE
>30 POP D,M,Y : REM GET THE DATA FROM THE ARGUMENT STACK
>40 PRINT “CURRENT DATE IS ”,Y,M,D
>50 END

READY
>RUN

CURRENT DATE IS  91  6  19

READY

>
```

CALL 45 – Retrieve Time String

Purpose

Use CALL 45 to return the current time in a string (HH:MM:SS). PUSH the number of the string to receive the time. You must allocate a minimum of 8 characters for the string.

Syntax

PUSH [string number]
CALL 45

Example

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1 : CALL 45 : REM PUT TIME IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

06:40:49

READY
>
```

CALL 46 – Retrieve Time Numeric

Purpose

Use CALL 46 to return the time of day in numeric form. Retrieve the time of day in numeric form by executing CALL 46 and POPping the three variables off of the argument stack on return. There are no input arguments. The time is POPped in hour, minute, and second order.

Syntax

CALL 46
POP [hour]
POP [minute]
POP [second]

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM TIME IN VARIABLES EXAMPLE : REM GET THE WALL CLOCK
    TIME
>20 CALL 46
>30 POP H,M,S
>40 PRINT  “CURRENT TIME IS ”,H,M,S
>50 END

READY
>RUN

CURRENT TIME IS  6  43  7

READY
>
```

CALL 47 – Retrieve Day of Week String

Purpose

Use CALL 47 to return the current day of week as a 3-character string. PUSH the number of the string to receive the day of week. You must allocate a minimum of 3 characters per string. Strings returned are SUN, MON, TUE, WED, THU, FRI, and SAT.

Syntax

PUSH [string number]
CALL 47

Example

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
```

```
>20 PUSH 0 :CALL 47
>30 PRINT "TODAY IS ",$(0)

READY
>RUN

TODAY IS FRI

READY
>
```

CALL 48 – Retrieve Day of Week Numeric

Purpose

Use CALL 48 to return the current day of week on the argument stack as a number (example: **`Sunday=1, Saturday=7`**). This number can be POPped into a variable.

Syntax

CALL 48
POP [day of week]

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM DAY OF WEEK RETRIEVE - NUMERIC EXAMPLE
>20 CALL 48 : REM INVOKE UTILITY TO GET D.O.W.
>30 POP D
>40 PRINT D
>50 END

READY
>RUN

 5

READY
>
```

CALL 52 – Retrieve Date String

Purpose

Use CALL 52 to return the current date in a string (dd-mmm-yy). PUSH the number of the string to receive the date. You must allocate a minimum of 9 characters for the string.

Syntax

PUSH [string number]
CALL 52

Example

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1 : CALL 52 : REM PUT DATE IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

16-JUN-91

READY
>
```

Status Functions

This chapter describes and illustrates commands that monitor the status of the BASIC or BASIC-T module. This chapter also describes and illustrates commands that allow the setup of the DF1 driver within the BASIC program or from the command line. Table 11.1 lists the corresponding mnemonics.

Table 11.1 Chapter Reference Guide

If you need to	Use this mnemonic	Page
Get the number of characters in PRT2 buffers.	CALL 36	11-2
Check the CPU output image buffer.	CALL 51	11-3
Check the CPU input image buffer.	CALL 55	11-4
Check the M0 file status.	CALL 58	11-5
Check the M1 file status.	CALL 59	11-6
Check the SLC 500 controller CPU status.	CALL 75	11-7
Check the battery condition.	CALL 80	11-8
Check the DH485 interface file remote Write status.	CALL 86	11-8
Check the DH485 interface file remote Read status.	CALL 87	11-9
Get the number of characters in the PRT1 buffers.	CALL 95	11-10
Enable port PRT2 DTR signal.	CALL 97	11-11
Disable port PRT2 DTR signal.	CALL 98	11-11
Enable DF1 driver communications.	CALL 108	11-12
Disable DF1 driver communications.	CALL 113	11-18
Clear the module input and output buffers.	CALL 120	11-18
Get the SLC processor program ID number.	CALL 121	11-19

CALL 36 – Get Number of Characters in PRT2 Buffers

Purpose

Use CALL 36 to retrieve the number of characters in the chosen buffer of port PRT2.

You must PUSH the buffer that you want examined:

PUSH 1 for the input buffer

PUSH 0 for the output buffer

One POP is required to get the number of characters.

Syntax

PUSH [buffer selection]
CALL 36
POP [number of characters]

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM EXAMINES THE OUTPUT 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

READY
>RUN

NUMBER OF CHARACTERS IN OUTPUT BUFFER IS 0

READY
>
```

CALL 51 – Check CPU Output Image Buffer

Purpose

Use CALL 51 to determine if the SLC 500 controller output image buffer located in the module has been updated since the last time it was checked. (In this case, update means that the data was written to these buffers from the CPU, even if the data is the same value.) This routine has no input arguments and one output argument.

The output argument is equal to:

- 0 – if the Logic Processor has not written to the output image buffer since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 1 – if the Logic Processor has written to the output image buffer since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 2 – if the Logic Processor does not support this capability (as with the SLC 5/01 processor)

Syntax

CALL 51
POP [output image buffer status]

Example

```
>1   REM EXAMPLE PROGRAM
>120 CALL 51 : REM WAIT ON SLC
>130 POP S
>140 IF (S = 2) THEN PRINT “THE SLC DOES NOT SUPPORT THIS
     FUNCTION”
>150 IF (S = 2) THEN STOP
>160 IF (S = 0) THEN GOTO 120
>170 PRINT “OUTPUT IMAGE HAS BEEN UPDATED”

READY
>RUN

THE SLC DOES NOT SUPPORT THIS FUNCTION
STOP - IN LINE 160
READY
```

CALL 55 – Check CPU Input Image Buffer

Purpose

Use CALL 55 to determine if the SLC 500 controller input image buffer located in the module has been read by the Logic Processor since the last time it was checked. This routine has no input arguments and one output argument.

The output argument is equal to:

- 0 – if the Logic Processor has not read from the input image buffer since the last time the CALL was executed or since the module was powered up, whichever occurred last
- 1 – if the Logic Processor has read from the input image buffer since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 2 – if a Logic Processor does not support this capability (as with the SLC 5/01 processor)

Syntax

CALL 55
POP [input image buffer status]

Example

```
>1   REM EXAMPLE PROGRAM
>120 CALL 55 : REM WAIT ON SLC
>130 POP S
>140 IF (S=2) THEN PRINT “THE SLC DOES NOT SUPPORT THIS
     FUNCTION”
>150 IF (S=2) THEN STOP
>160 IF (S=0) THEN GOTO 120
>170 PRINT “INPUT IMAGE HAS BEEN READ”

READY
>RUN

INPUT IMAGE HAS BEEN READ

READY
>
```

CALL 58 – Check M0 File

Purpose

Use CALL 58 to determine if the Module File M0 located in the module has been updated since the last time it was checked. (In this case, update means that the data was written to these buffers from the CPU, even if the data is the same value.) This routine has no input argument and one output argument.

The output argument is equal to:

- 0 – if the Logic Processor has not written to the Module File M0 since the last time this CALL was executed or since the module was powered up
- 1 – if the Logic Processor has written to the Module File M0 since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 2 – if the Logic Processor does not support this capability (as with the SLC 5/01 processor)

Syntax

CALL 58
POP[module file M0 write status]

Example

```
>1   REM EXAMPLE PROGRAM
>120 CALL 58 : REM START WAITING ON M0 UPDATE
>130 POP S
>140 IF (S = 2) THEN PRINT “PROCESSOR DOES NOT SUPPORT THIS
FUNCTION”
>150 IF (S = 2) THEN STOP
>160 IF (S = 0) THEN GOTO 120
>170 PUSH 64
>180 CALL 56
>190 POP A
>200 PRINT “CALL 56 OUTPUT IS ”,A


READY
>RUN

CALL 56 OUTPUT IS 0
```

CALL 59 – Check M1 File

Purpose

Use CALL 59 to determine if the Module File M1 located in the module has been read by the Logic Processor since the last time it was checked. This routine has no input arguments and one output argument.

The output argument is equal to:

- 0 if the Logic Processor has not read from the Module File M1 since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 1 if the Logic Processor has read from the Module File M1 since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 2 if the Logic Processor does not support this capability (as with the SLC 5/01 processor)

Syntax

CALL 59
POP [module file M1 read status]

Example

```
>1   REM EXAMPLE PROGRAM
>100 PUSH 64
>110 CALL 56 : REM COPY BASIC OUTPUT BUFFER TO M1
>120 POP A
>130 IF (A=2) THEN PRINT “SLC DOES NOT SUPPORT THIS
     FUNCTION”
>140 IF (A=2) THEN STOP
>150 IF (A<>0)THE GOTO 110
>160 CALL 59 : REM START WAITING NOW
>170 POP S
>180 IF (S=2) THEN PRINT “SLC DOES NOT SUPPORT THIS
     FUNCTION”
>190 IF (S=2)THE STOP
>200 IF (S=0) THEN GOTO 170
>210 PRINT “CALL 59 OUTPUT IS ”,S

READY
>RUN

CALL 59 OUTPUT IS 1
```

CALL 75 – Check SLC 500 Controller CPU Status

Purpose

Use CALL 75 to check the mode (Run/Program/Test) of the SLC processor. No PUSHes are required. One POP is required.

In SLC 5/01 mode of operation, the POPped values are:

- 0 - SLC processor in Run mode
- 1 - SLC processor not in Run mode

In SLC 5/02 mode of operation, the POPped values are:

- 0 - SLC processor in Run mode
- 1 - SLC processor in Program mode
- 2 - SLC processor in Test mode

Syntax

CALL 75
POP [processor mode]

Example

```
>1   REM EXAMPLE PROGRAM
>100 CALL 75
>110 POP S
>120 IF (S=0) THEN PRINT “SLC IS IN RUN MODE”
>130 IF (S=1) THEN PRINT “SLC IS NOT IN RUN MODE”
>140 IS (S=2) THEN PRINT “SLC IS IN TEST MODE”

READY
>RUN

SLC IS IN RUN MODE

READY
>
```

CALL 80 – Check Battery Condition

Purpose

Use CALL 80 to check the module battery condition. If a 0 is POPped after a CALL 80, the battery is okay. If a 1 is POPped a low battery condition exists.

Syntax

CALL 80
POP [battery status]

Example

```
>1 REM EXAMPLE PROGRAM
>10 CALL 80
>20 POP C
>30 IF (C<>0) THEN PRINT ”BATTERY LOW!”
>40 END

READY
>RUN

BATTERY LOW!
```

CALL 86 – Check DH485 Interface File Remote Write Status

Purpose

Use CALL 86 to determine if the DH485 Common Interface File located in the module has been updated since the last time it was checked. This routine has no input arguments and one output argument.

The output argument is equal to:

- 0 if a device on the DH485 Serial Communications Link has not written to the DH485 Serial Common Interface File since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 1 if a device on the DH485 Serial Communications Link has written to the DH485 Common Interface File since the last time this CALL was executed or since the module was powered up, whichever occurred last

Syntax

CALL 86
POP [DH485 interface file remote write status]

Example

```
>1   REM EXAMPLE PROGRAM
>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

READY
>
```

CALL 87 – Check DH485 Interface File Remote Read Status

Purpose

Use CALL 87 to determine if the DH485 Common Interface File located in the module has been read by a device on the DH485 Serial Communications Link since the last time it was checked. This routine has no input arguments and one output argument.

The output argument is equal to:

- 0 if a device has not read from the DH485 Common Interface File since the last time this CALL was executed or since the module was powered up, whichever occurred last
- 1 if a device on the DH485 Serial Communications Link has read the DH485 Common Interface File since this CALL was executed or since the module was powered up, whichever occurred last

Syntax

CALL 87
POP [DH485 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

READY
>
```

CALL 95 – Get Number of Characters in PRT1 Buffers

Purpose

Use CALL 95 to retrieve the number of characters in the chosen buffer of port PRT1.

You must PUSH which buffer you want examined:

- PUSH 1 for the input buffer
- PUSH 0 for the output buffer

One POP is required to get 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 OUTPUT BUFFER
>20 CALL 95
>30 POP X : REM GET THE NUMBER OF CHARACTERS
>40 PRINT “NUMBER OF CHARACTERS IN PRT1 OUTPUT BUFFER IS ”,X
>50 END

READY
>RUN

NUMBER OF CHARACTERS IN PRT1 OUTPUT BUFFER IS 0

READY
>
```

CALL 97 – Enable Port PRT2 DTR Signal

Purpose

Use CALL 97 to enable the Data Terminal Ready (DTR) signal from port PRT2. The DTR signal on PTR2 is enabled by default on powerup. This CALL re-enables the DTR signal if it has been disabled by CALL 98.

Syntax

CALL 97

Example

```
>10 REM EXAMPLE PROGRAM
>20 CALL 97 : REM ENABLE DTR SIGNAL


READY
>
```

CALL 98 – Disable Port PRT2 DTR Signal

Purpose

Use CALL 98 to disable the Data Terminal Ready (DTR) signal from port PRT2.

Syntax

CALL 98

Example

```
>10 REM EXAMPLE PROGRAM
>20 CALL 98 : REM DISABLE DTR SIGNAL


READY
>
```

CALL 108 – Enable DF1 Driver Communications

Purpose

Use CALL 108 to enable DF1 driver communications via port PRT2.

IMPORTANT DF1 can only be enabled if jumper JW4 on the module is in the correct position. Refer to the *SLC 500™ BASIC and BASIC-T Modules User Manual* (publication number 1746-UM004A-US-P) for additional information.

This routine has six input arguments and no output arguments. 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). If ADER is selected, enabled responses are not transmitted until an embedded response is received. It is assumed that if the device being communicated with can send an ER it can also receive them (only applies to full-duplex operation).
- modem handshaking selection:
  - half-duplex options:
    - No HandShaking (NHS)
    - Half-Duplex Modem without Continuous Carrier (HDMwoCC)
    - Half-Duplex Modem with Continuous Carrier (HDMwCC)
  - full-duplex options:
    - No HandShaking (NHS)
    - Full-Duplex Modem (FDM)

Legal values for the operational code are 0 to 11 for half-duplex mode and 16 to 31 for full-duplex mode. Table 11.B lists the legal values for the half-duplex operational codes and their corresponding mode of operation. Table 11.C lists the legal values for the full-duplex operational codes and their corresponding mode of operation.

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.

IMPORTANT Other port parameters, such as baud rate, number of stop bits, and parity are selected using the MODE command before DF1 is enabled. The modem handshaking selection made here overrides the handshaking parameter of the MODE command until DF1 is disabled.

Table 11.2 DF1 Half-Duplex Operational Codes

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

Table 11.3 DF1 Full-Duplex Operational Codes

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

Half-Duplex No Handshaking Modem Control

Half-duplex no handshaking modem control, selected by operational codes 0 through 3, has the following characteristics:

- The RTS output line is activated during transmission, but no RTS On Delay or RTS Off Delay is performed.
- The 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
- The CTS and DSR input lines are *not* monitored nor do they have any affect on transmissions or receptions.
- A transmission monitor guarantees that transmitter interrupts are being generated in a timely manner. If a timeout occurs, the DF1_Status is set to code value 5 if a data packet was being transmitted. Also, RTS is immediately dropped when this timeout occurs.

Half-duplex without continuous carrier modem control, selected by operational codes 4 through 7, has the following characteristics:

IMPORTANT For proper operation, the Data Carrier Detect (DCD) line from the modem must be connected to the DSR input of port PRT2.

- The 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.
- An actual transmission does not start until the CTS input is active. A transmission guarantees that transmitter interrupts are being 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. Even after the DF1 Driver is disabled, it *will* remain active; the user may deactivate it via a BASIC Call.
- Characters that are 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 Carrier Modem Control

Half-duplex with continuous carrier modem control, selected by operational codes 8 through 11, has the following characteristics:

IMPORTANT For proper operation, the Data Carrier Detect (DCD) line from the modem must be connected to the DSR input of port PRT2.

- The 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.
- An actual transmission does not start until the CTS input is active. A transmission guarantees that transmitter interrupts are being 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 as described in the next paragraph. Even after the DF1 Driver is disabled or remains active, the user may deactivate it via a BASIC CALL.
- 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. 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.

Full-Duplex With No Handshaking

Full-duplex with no handshaking, selected by operation codes 16 through 23, has the following characteristics:

- The RTS output line is activated when the DF1 Driver is enabled and remains so until the DF1 Driver is disabled.
- The DTR output line is not manipulated by the DF1 Driver. It is recommended that the user activate DTR in his/her BASIC program while the DF1 communications is taking place.
- The CTS and DSR input lines are *not* monitored or have any effect on transmissions.
- A transmission monitor guarantees that transmitter interrupts are being generated in a timely manner. If a timeout occurs, then the DF1_Status is sent to code value 5 if the packet in the process of being transmitted was a data packet. RTS is *not* deactivated when this timeout occurs.

Full-Duplex Modem (FDM)

Full-Duplex Modem (FDM), selected by operation codes 24 through 31, has the following characteristics:

- The 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 being generated in a timely manner. If a timeout occurs, then the DF1_Status is sent to code value 5 if the packet in the progress of being transmitted was a data packet. RTS is *not* 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; the user may deactivate it via a BASIC CALL.
- 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.

  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.

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 Timeout 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 Timeout specifies in 5 ms increments how long to wait for an ACK/NAK before transmitting an ENQuiry. The valid range for the ACKnowledge Timeout is 2 to 65535.

The third input argument specifies the number of message retries when in half-duplex mode or the number of ENQuiry 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. ENQuiry 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.

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.

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.

The sixth input argument specifies the 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 [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 module ADDRESS OF 10
>70 CALL 108
>80 END
```

CALL 113 – Disable DF1 Driver Communications

Purpose

Use CALL 113 to disable DF1 driver communications. This routine has no input arguments and no output arguments. 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 transmission and receive buffers.

Syntax

CALL 113

Example

```
>1 REM EXAMPLE PROGRAM
>10 CALL 113
>20 END
```

CALL 120 – Clear module Input and Output Buffers

Purpose

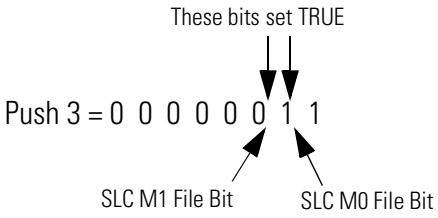
Use CALL 120 to clear the module input and output buffers. This routine has one input argument and no output arguments. The input argument is an 8-bit word

that corresponds to the module input and output buffers, as shown in the table below:

Table 11.4 Information for Clearing Input and Output Buffer

Bit	Decimal Equivalent	Module Input and Output Buffer Areas
0	1	SLC M0 File
1	2	SLC M1 File
2	4	SLC Output Image Table
3	8	SLC Input Image Table
4	16	Common Interface Input File
5	32	Common Interface Output File
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 SLC M0 and SLC M1 files you would PUSH the value 3. The module sets bits 0 and 1 true and clears the SLC M0 and M1 file areas of the input and output buffers.



Syntax

PUSH [decimal equivalent]
CALL 120

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 3 : REM CLEAR SLC MO AND M1 FILES
>20 CALL 120
>30 END
```

CALL 121 – Get SLC Processor Program ID Number

Purpose

Use CALL 121 to get the ID number of the active SLC processor program. This routine has no input arguments and one output argument. The output argument is an integer value between 0 and 65536 that corresponds to the program ID number of the active program on the SLC processor.

Syntax

CALL 121
POP [program ID number]

Example

```
>1  REM EXAMPLE PROGRAM
>10 CALL 121
>20 POP X : REM GET SLC PROCESSOR PROGRAM ID
>30 END
```

Output Functions

This chapter describes and illustrates commands that allow the transfer of data from the BASIC or BASIC-T module to external ports PRT1, PRT2, and DH485 within the BASIC program or from the command line. Table 12.1 lists the corresponding mnemonics.

Table 12.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Transfer data from the SLC I/O or M files to PRT1 or PRT2.	CALL 23	12-2
Transfer data from the SLC processor to a remote DH485 data file.	CALL 28	12-6
Handle all errors that are not handled by the ONERR statement.	CALL 29	12-13
Display the current PRT2 port setup.	CALL 31	12-14
Clear port PRT2 input and output buffers.	CALL 37	12-15
Transfer module output buffer to the CPU input image buffer.	CALL 54	12-15
Transfer module output buffer to the CPU M1 file.	CALL 57	12-16
Transfer module output buffer to the DH485 interface file.	CALL 85	12-17
Write module output buffer to the remote DH485 data file.	CALL 91	12-18
Write module output buffer to the remote DH485 interface file.	CALL 93	12-22
Print current PRT1 port setup.	CALL 94	12-24
Clear port PRT1 input and output buffers.	CALL 96	12-24
User LED control	CALL 112	12-25
Transmit the DF1 packet.	CALL 114	12-26
Check the DF1 XMIT status.	CALL 115	12-27
Write to a PLC data file.	CALL 123	12-28
Print hex value with zero suppression to the console device.	PH0.	12-37
Print hex value with zero suppression to PRT2.	PH0.#	12-37
Print hex value with zero suppression to PRT1.	PH0.@	12-37
Print hex value with no zero suppression to the console device.	PH1.	12-37
Print hex value with no zero suppression to PRT2.	PH1.#	12-37
Print hex value with no zero suppression to PRT1.	PH1.@	12-37
Print variables, strings, or literals to the console device; P. is shorthand for print.	PRINT	12-35
Print to port PRT2.	PRINT#	12-35
Print to port PRT1.	PRINT@	12-35
Print carriage return.	PRINT CR	12-36
Print spaces.	PRINT SPC( )	12-36
Print tabs.	PRINT TAB( )	12-36
Print numeric values in scientific notation.	PRINT USING(Fx)	12-36

Table 12.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Print numeric values in decimal notation.	PRINT USING(#.#)	12-36
Restore the default print mode.	PRINT USING(0)	12-37
Store variable.	ST@	12-38

CALL 23 – Transfer Data from the CPU Files to Port 1 or 2

Purpose

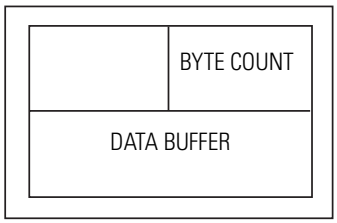
Use CALL 23 to transfer data from the CPU output data file or the CPU M0 file directly to the module serial port and/or to a string within the module. The data is transferred low byte first, then high byte or high byte first, then low byte to the module port. The data can also be stored in a string for access by the BASIC program.

The byte swap selection (low byte first, then high byte, or high byte first, then low byte) of the last CALL 23 or CALL 22 executed determines the data packing method for all ports enabled by CALL 23.

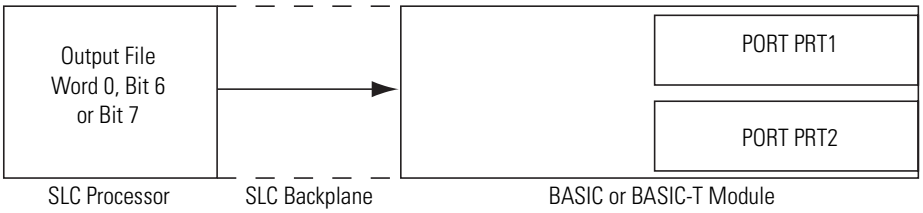
The low byte of the first word of the source file contains the character count (byte count) of the data being transferred. If the byte count is larger than the file selected, only the maximum number of bytes within the file are transferred. The high byte of the first word is not used.

Execute CALL 23 to set up the data transfer parameters. After the CALL is executed, the module gets data from the source file and transfers it to port PRT1, port PRT2, or an internal string. Input and Output image file bits (word 0, bits 6 and 7) for the slot containing the module, are used to initiate and notify completion of the transfer. The operation is described below:

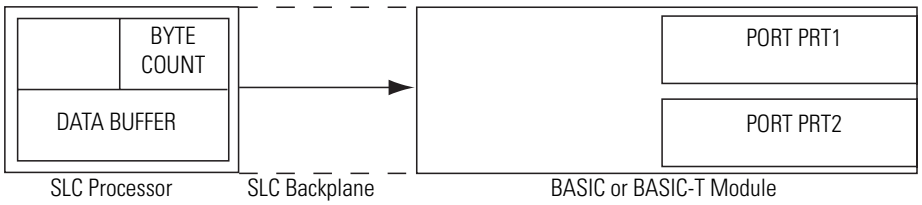
1. The ladder logic program of the SLC processor builds the data buffer. The SLC then determines the byte count of the file to be transferred and places it into the lower byte of the first available word to be transferred. This word plus the data comprise the data file to be transferred.



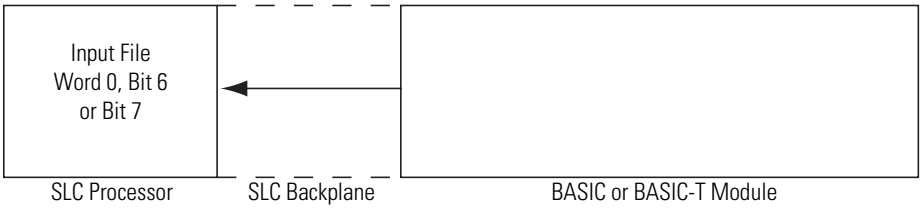
2. The ladder logic program of the SLC processor must set the output file word 0, bit 6 or bit 7 to inform the module that valid data is available. Bit 6 indicates that data is available for port 1 and bit 7 indicates that data is available for port 2.



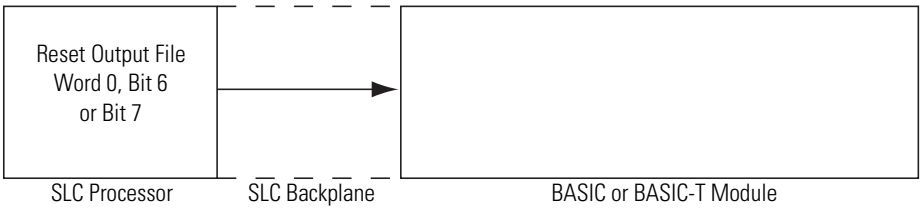
3. The module automatically transfers the data to the destination serial port (PRT1 or PRT2) from the SLC processor.



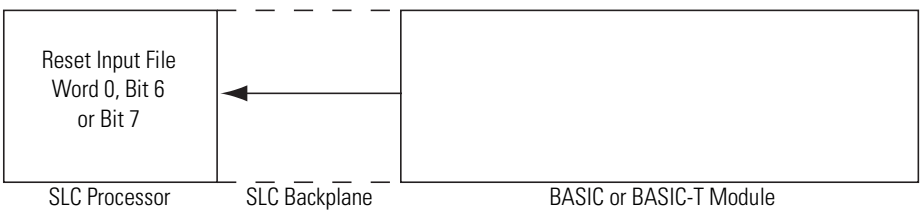
4. The module sets the input file word 0, bit 6 or bit 7 to inform the SLC processor that the transfer was successful.



5. The ladder logic program of the SLC processor resets output file word 0, bit 6 or bit 7.



6. The module resets the input file word 0, bit 6 or bit 7 on the same end of scan cycle in which the output file word 0, bit 6 or bit 7 was reset.



Transfers continue in this manner until the CALL for the port is re-executed with different input parameters. If this occurs, the previous CALL 23 for the port is automatically disabled and the new CALL 23 takes effect. Multiple CALL 23s for the same port are not executed in parallel. However, port 1 and port 2 can be activated simultaneously by issuing separate CALL 23s for these ports.

This CALL has five input arguments and one output argument.

The first input argument chooses the destination of the data. It can be the port number (1 or 2) and/or the internal string:

- 0 - Disable CALL 23 for all active ports and strings enabled by earlier CALL 23s
- 1 - Internal string only
- 2 - Serial port 1
- 3 - Internal string and serial port 1
- 4 - Serial port 2
- 5 - Internal string and serial port 2

If an internal string (1, 3, or 5) is chosen, the first character of the string contains the byte count. The second character (transaction number) is incremented to inform the 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.

The second input argument is the selection of the data source. It can be the CPU output data file or CPU M0 file:

- 0 - CPU output image file
- 1 - CPU M0 file

The third input argument is the word offset within the CPU output image file or M0 file. This offset points to the first word, which contains the byte count of the valid data in the file. The second word of the CPU file contains the first two characters of the data to be transferred. If the CPU output image file is chosen, this offset must not be word 0 since word 0 is reserved for the handshaking bits used in this CALL. Therefore, the first available word for the byte count when the output image is chosen is the second word (word 1).

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 must still be PUSHed). If the data exceeds the string length, the remaining data is truncated.

The fifth input argument is the byte swap selection. It has the following values:

- 0 - Data bytes transferred from the CPU are *not* swapped when passed to the 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 CPU are swapped when passed to the 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 23 executed determines the byte swap option for all active CALL 23 and CALL 22 commands previously executed.

The output argument is the status of the CALL. It has the following values:

- 0 - Successful
- 1 - Disabled
- 2 - Bad input parameter
- 3 - PRT2 is chosen but it is already enabled for DF1 protocol. The CALL is not executed.
- 4 - String is too small
- 5 - String is not dimensioned

Data transfer does not begin until the SLC processor sets output image file word 0, bit 6 or bit 7 (depending on the destination port). The module sets the input file word 0, bit 6 or bit 7 to indicate a successful transfer to a port.

If the internal string is chosen as the destination, output image file word 0, bit 6 is used to initiate the transfer. Input file word 0, bit 6 is set by the module to indicate a successful transfer to the string.

Syntax

PUSH [destination port number and/or internal string]

PUSH [selection of source file]

PUSH [word offset within the source file]

PUSH [string number]

PUSH [byte swap selection]

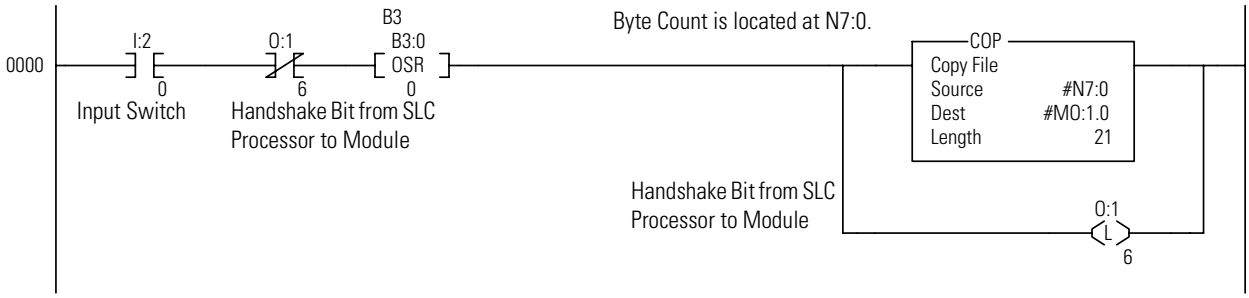
CALL 23

POP [CALL 23 status]

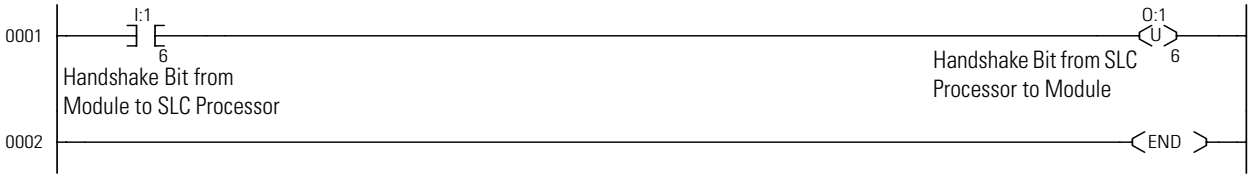
Example

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE CALL 23 INTERRUPTS
>20 PUSH 2 : REM SEND DATA TO PRT1
>30 PUSH 1 : REM GET DATA FROM M0 FILE
>40 PUSH 0 : REM WORD OFFSET INTO M0 FILE
>50 PUSH 0 : REM STRING NUMBER/NOT USED HERE
>60 PUSH 1 : REM ENABLE BYTE SWAPPING
>70 CALL 23
>80 POP S : REM STATUS OF CALL SETUP
>90 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 23 SETUP"
```

Below is a sample ladder logic program for CALL 23. The module is located in slot 1 of the SLC chassis. At rung 0000, copy data to the M0 file and set the handshake bit (O:1.0/6) to inform the module that new data is available for port PRT1. Do not send more data until the previous data has been acknowledged by the module. The data file for the module begins with the byte count (40 in this example) at N7:0. The data follows in N7:1-N7:20. The data byte count word is not included in the data byte count. The copy instruction word length is 21. The byte count maximum is 40 bytes (20 words) for this example.



At rung 2:1, unlatch the handshake bit from the SLC (O:1.0/6) for port PRT1 when the handshake bit from the module is set indicating the module has received the data.



CALL 28 – Write to Remote DH485 SLC Data File

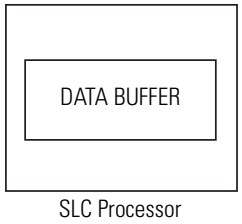
Purpose

Use CALL 28 to write up to 40 words of data from the CPU output image file, CPU M0 file, and/or a string within the module to the remote DH485 file at the designated node address.

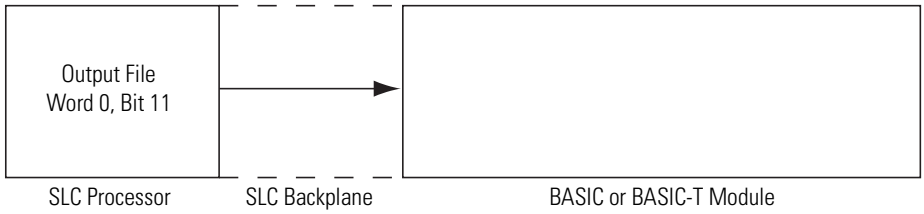
If an internal string is chosen, the first character (transaction number) is incremented on successful completion of the write to inform the module that data was sent. The value of the transaction number wraps around from 255 to 0.

Execute CALL 28 once to set up the data transfer parameters. Input and output image bits (word 0, bit 11) for the slot containing the module, are used to initiate and notify completion of the transfer. The operation is described below:

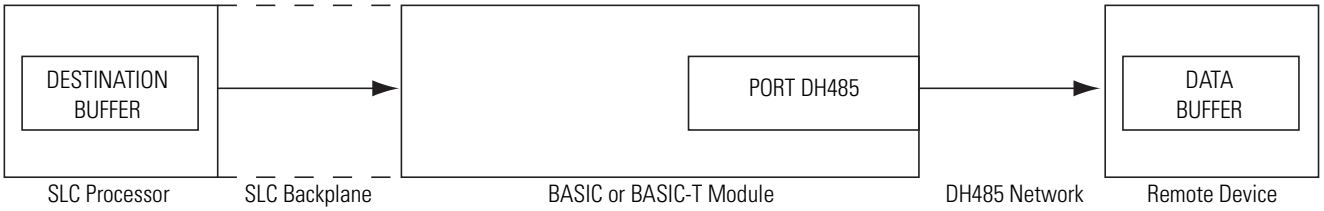
1. The local SLC processor builds the data buffer.



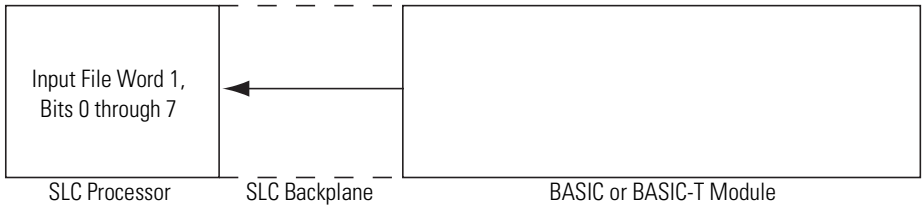
2. The SLC processor sets output file word 0, bit 11 to inform the module that data can be transferred.



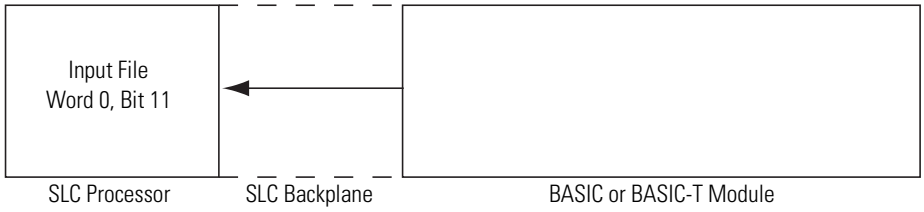
3. The module transfers the data into the remote data buffer.



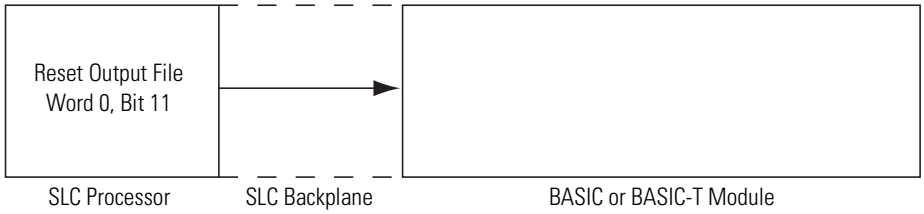
4. The module places the transfer status into the input file word 1, bits 0-7.



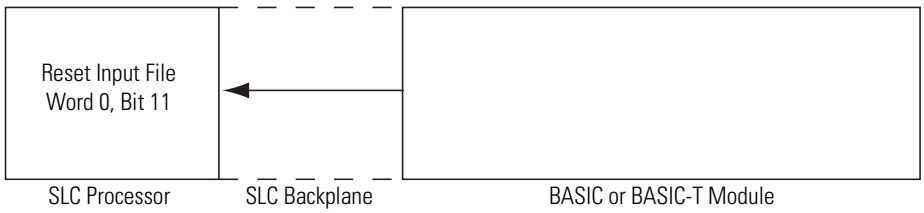
5. The module sets the input file word 0, bit 11 to inform the local SLC processor that valid data is transferred and the status of the transfer is available.



6. The SLC retrieves the status and resets output file word 0, bit 11. The data buffer can be reused by the SLC.



7. The module resets the input file word 0, bit 11 on the same end of scan cycle in which the output file word 0, bit 11 was reset.



Once this CALL is active, it will remain active and will send data to the remote node whenever the SLC processor sets output file word 0, bit 11.

This CALL has ten input arguments and one output argument.

The first input argument is the type of SLC WRITE command issued:

- 0 - Disable the previously executed CALL 28
- 1 - Common Interface File Write
- 2 - SLC Typed Write

The second input argument is the node address of the SLC remote device on the DH485 network (0 through 31). If the number is not within this range, the status equals 2 and the write message does not occur.

The third input argument is the file number on the SLC remote device (0 through 255). If the number is not within this range, the status equals 2 and the write message does not occur. The parameter is ignored if the Common Interface File is chosen in the first parameter. The CIF is always file 9.

The fourth input argument is the file type to be written to 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 write message does not take place. Enter the file type code as shown:

Table 12.2 FileType

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

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.

The sixth input argument is the number of words to be transferred. If the number is not within the range shown, the status equals 2 and the transfer does not occur.

Table 12.3 Valid Length Range

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

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, bits 0-7. If the time-out value is not within the range (1 through 255), the status POPped equals 2 and the transfer does not take place.

The eighth input argument is the selection of the source CPU output image file, CPU M0 file, or the internal string:

- 0 - CPU output image file
- 1 - CPU M0 file
- 2 - Internal string

If you chose internal string (2), CALL 29 can be executed to initiate each data transfer without requiring SLC processor interaction. The output file word 0, bit 11 will also initiate a string transaction.

The ninth input argument is the word offset within the source CPU file. The word offset points to the first data location. If you chose the CPU output image file, the

offset should not be 0 since this word is reserved for the data transfer handshaking bits. The offset for the internal string is always 1. The string character (transaction number) at location 0 is incremented upon every data transfer.

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 must still be PUSHed.

The output argument is the validation of the CALL parameters. It has the following values:

- 0 - Successful
- 1 - Disabled
- 2 - Bad input parameter
- 3 - Port DH485 not enabled (DF1 enabled)
- 4 - String is too small
- 5 - String is not dimensioned
- 6 - Data packet is too long

To disable this CALL, a zero must be PUSHed into the first input parameter. All other parameters are ignored but must still be PUSHed.

Whenever an attempt is made to write to a remote node, the module places the status of the write into the SLC input file word 1, bits 0-7. The status values have the same definition as the values POPped in CALL 93.

Syntax

PUSH [type of WRITE command]

PUSH [remote node address]

PUSH [remote file number]

PUSH [remote file type]

PUSH [remote starting word offset]

PUSH [number of words to be transferred]

PUSH [message time-out value]

PUSH [selection of source file]

PUSH [word offset within source file]

PUSH [string number]

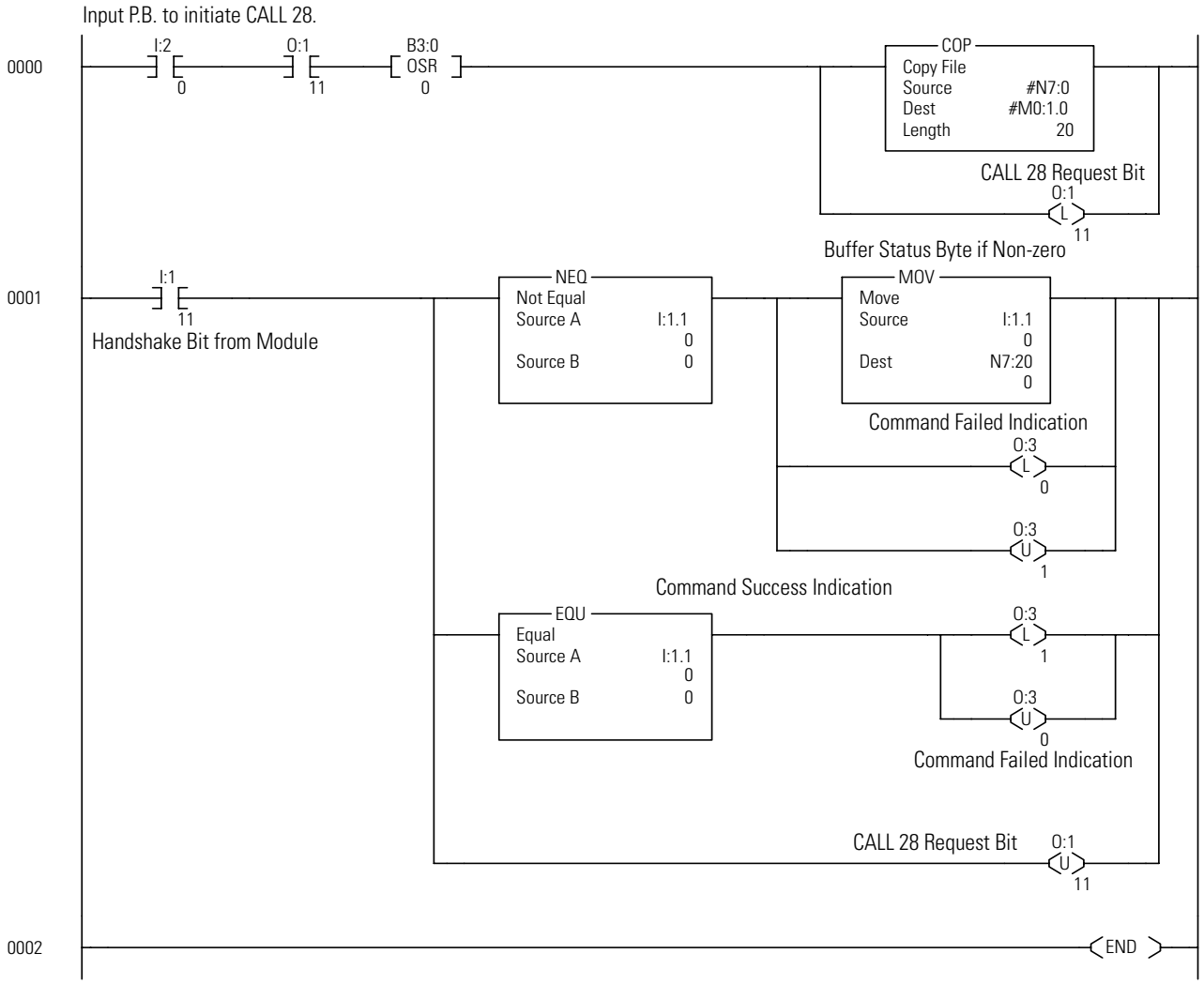
CALL 28

POP [CALL 28 status]

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE INTERRUPTS FOR READ COMMANDS TO A
     REMOTE DH485 NODE
>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 1 : REM LOCAL SLC FILE TYPE (M0)
>100 PUSH 0 : REM WORD OFFSET INTO LOCAL SLC FILE
>110 PUSH 0 : REM INTERNAL STRING NUMBER - NOT USED FOR THIS
EXAMPLE
>120 CALL 28
>130 POP S
>140 IF (S=1) THEN PRINT “CALL 28 DISABLED”
>150 IF (S=2) THEN PRINT “CALL 28 BAD INPUT PARAMETER”
>160 IF (S=3) THEN PRINT “PORT DH485 NOT ENABLED”
```

Copy the data to be sent to the remote DH485 node to the local file (the M0 file associated with the slot number of the module). Latch the CALL 28 request bit (word 0, bit 11).


Input P.B. to initiate CALL 28.
0000
I:2
0
O:1
11
B3:0
OSR
0
COP
Copy File
Source #N7:0
Dest #M0:1.0
Length 20
CALL 28 Request Bit
O:1
L
11
Buffer Status Byte if Non-zero
0001
I:1
11
Handshake Bit from Module
NEQ
Not Equal
Source A I:1.1
0
Source B 0
MOV
Move
Source I:1.1
0
Dest N7:20
0
Command Failed Indication
O:3
L
0
O:3
U
1
Command Success Indication
EQU
Equal
Source A I:1.1
0
Source B 0
O:3
L
1
O:3
U
0
Command Failed Indication
CALL 28 Request Bit
O:1
U
11
0002
END

CALL 29 – Read/Write to a PLC/SLC from the Module Internal String

Purpose

Use CALL 29, in conjunction with CALLs 122 or 123, to communicate between remote PLCs and the module internal string without local SLC processor interaction. You can also use CALL 29, in conjunction with CALLs 27 or 28, to communicate between remote SLCs and the module internal string without local SLC processor interaction. CALLs 27, 28, 122, or 123 must be executed within the BASIC program before CALL 29.

CALL 29 is active when the internal string is the only choice in CALLs 27, 28, 122, or 123. In this situation, it is not practical to use the SLC input and output image files to begin the transfer and to pass the status. The SLC processor does not need to be involved. If an SLC file is chosen instead, the local SLC processor controls the transfer with the input and output image bits. In this instance, a status of 255 is returned when CALL 29 is attempted.

One argument is PUSHed and one argument is POPped. The input argument is the CALL to be activated (CALL 27, 28, 122, or 123).

When CALL 29 is executed, the transfer is attempted. If the selected CALL (27, 28, 122, or 123) is not executed before CALL 29, a 1 is returned in the status POPped. When CALL 29 is executed successfully, the value in the first character of the string (transaction number) is incremented to designate that the transfer occurred. The range of this character is 0-255.

After CALL 29 is executed, one word is POPped. This word is the status of the transaction:

- 0 - Successful completion
- 1 - The chosen CALL (27, 28, 122, or 123) is not active
- 255 - SLC buffer is chosen for CALL 27, 28, 122, or 123 and CALL 29 is ignored
- All other codes are identical to CALL 90/92

Syntax

PUSH [27, 28, 122, or 123 for the CALL you want activated]

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® at node 3, to the internal string $(1) of the module.

```
>1   REM EXAMPLE PROGRAM
>10  REM EXECUTE DF1 PLC REMOTE READ FROM INTERNAL
>20  REM STRING WITH NO SLC INTERVENTION
>21  REM SET UP CALL 122
>25  PUSH 5, 3, 10, ASC(N), 0, 10, 10, 1, 1, 1: CALL 122: 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 “SLC FILE CHOSEN FOR CALL 122”
>80  IF (S=0) THEN PRINT “SUCCESSFUL TRANSFER”
>90  IF (S<>0) THEN PRINT “UNSUCCESSFUL TRANSFER”
>100 END
```

CALL 29 does not require SLC bit handshaking at the end of the command, unlike CALLs 27, 28, 122, and 123 when using an SLC file as a source or destination.

CALL 31 – Display Current PRT2 Port Setup

Purpose

Use CALL 31 to display the current PRT2 port configuration on the program port terminal screen. No arguments are PUSHed or POPped.

Syntax

CALL 31

Example

```
>CALL 31

1200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff

READY
>
```

CALL 37 – Clear PRT2 Input/Output Buffers

Purpose

Use CALL 37 to clear the peripheral port input and/or output buffers. Use the following PUSHes to clear the corresponding buffer:

- PUSH 0 to clear the output buffer.
- PUSH 1 to clear the input buffer.
- PUSH 2 to clear both buffers.

Syntax

PUSH [buffer selection]
CALL 37

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM CLEARS THE OUTPUT BUFFER
>20 CALL 37
>30 END

READY
>RUN

READY
>
```

CALL 54 – Transfer BASIC Output Buffer to CPU Input Image

Purpose

Use CALL 54 to transfer words 200 to 207 of the module output buffer to words 0 to 7 of the CPU input image table. This routine has no input arguments and one output argument. The output argument is the status of the Logic Processor.

Word 200 in the BASIC output buffer is reserved and can not be modified. This word provides module status information to the SLC 500 processor. Bit 15 is the module mode bit. It can have one of the following values:

- 0 - BASIC or BASIC-T module is in the Run mode
- 1 - BASIC or BASIC-T module is not in the Run mode

Bit 14 of word 200 is the EEPROM checksum bit. It can have one of the following values:

- 0 - EEPROM checksum is correct
- 1 - EEPROM checksum is incorrect

Bit 13 of word 200 is the battery status bit. It can have one of the following values:

- 0 - Battery is good
- 1 - Battery is low

Word integrity is guaranteed during this transfer. File integrity is not. Handshaking bits can be used in your application program to provide file integrity.

Syntax

CALL 54
POP [processor mode]

Example

```
>1  REM EXAMPLE PROGRAM
>30 CALL 54 : REM XFER BASIC OUTPUT BUFFER TO CPU INPUT IMAGE
TABLE
>40 POP X : REM LOGIC PROCESSOR STATUS IS IN X
>50 IF X<>0 THEN PRINT “PROCESSOR NOT IN RUN MODE”

READY
>RUN

READY
>
```

CALL 57 – Transfer BASIC Output Buffer to CPU M1 File

Purpose

Use CALL 57 to transfer up to 64 words starting at word 100 from the module output buffer to the CPU M1 file starting at word 0. This routine has one input argument and one output argument. The input argument is the number of words to be transferred (1 to 64). If the number is not within the range 1 to 64, no transfer occurs and the output argument is set to 10.

If the input argument is a valid number, then the output argument is the status of the Logic Processor. It can have one of the following values:

- 0 - Successful transfer, SLC processor in Run mode
- 1- Successful transfer, SLC processor in Program mode
- 2 - Successful transfer, SLC processor in Test mode
- 10 - Illegal length specified
- 11 - SLC processor does not support this capability

Word integrity is guaranteed during this transfer. File integrity is not. Handshaking bits can be used in your application program to provide file integrity.

Syntax

PUSH [number of words to be transferred]
CALL 57
POP [processor mode]

Example

```
>1  REM EXAMPLE PROGRAM
>50 PUSH 64 : REM TRANSFER LENGTH IS 64 WORDS
>60 CALL 57 : REM TRANSFER BASIC OUTPUT BUFFER TO M1
>70 POP X : REM LOGIC PROCESSOR STATUS IS IN X
>80 PRINT X

READY
>RUN 0

READY
>
```

CALL 85 – Transfer BASIC Output Buffer to DH485 Common Interface File

Purpose

Use CALL 85 to transfer up to 40 words to the DH485 Common Interface File.

This routine has two input arguments and one output argument. The first input argument is the starting word offset of the DH485 Common Interface File. If the word offset value is not within the range 0 to 127, the output argument equals 1. The second input argument is the number of words to be transferred from the module output buffer to the DH485 Common Interface File. If the second input argument number is not within the range 1 to 40, the output argument equals 2.

The output argument specifies the transfer status. It can have one of the following values:

- 0 - Successful transfer
- 1 - Illegal starting offset
- 2 - Illegal length

Word integrity is guaranteed during this transfer. File integrity is not.

Syntax

PUSH [starting word offset in DH485 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 DH485 COMMON INTERFACE
    FILE
>70 POP R
>80 IF R<>0 PRINT "TRANSFER ERROR CODE = ",R : REM PRINT
    ERROR

READY
>RUN

READY
>
```

CALL 91 – Write BASIC Output Buffer to Remote DH485 Data File

Purpose

Use CALL 91 to write up to 40 words starting at word 0 of the module output buffer to the remote DH485 data file at the designated node address, file number, file type, and element offset. This routine has six input arguments and one output argument.

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 write message does not take place.

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.

The third input argument is the file type written to the remote device. Valid type codes are ASC(N), ASC(S), ASC(C), ASC(T), ASC(B), and ASC(R). If the file type is not one of these valid types, then the output argument equals 241, and the write message does not take place.

Table 12.4 FileType Written to the Remote Device

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

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 will be twice of what is expected. For example, if an offset of 3 was PUSHed, the data will be written to the remote DH485 data file beginning at element 6.

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.

Table 12.5 Valid Length Range

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

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 module output buffer written to the remote device starting at word 0 and filling as many words as specified by the element length of the message.

Upon return from the CALL, the output argument specifies the status of the message instruction. Table 12.6 defines the output argument.

Table 12.6 Output Argument

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	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	module detects illegal target node address.
11	0B	module detects illegal file number.
12	0C	module detects illegal target file element offset.
13	0D	module detects illegal target file length.
14	0E	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	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.

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

READY
>
```

CALL 93 – Write Output Buffer to Remote DH485 Common Interface File

Purpose

Use CALL 93 to write up to 40 words starting at word 0 of the module output buffer to the remote DH485 Common Interface File at the designated node address, starting at the designated word offset. This routine has four input arguments and one output argument.

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 write message does not take place.

The second input argument is the starting word 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 will be twice of what is expected. For example, if an offset of 3 was PUSHed, the data will be written to the remote DH485 data file beginning at element 6.

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.

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 module output buffer starting at word 0 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.

Upon return from the CALL, the output argument specifies the status of the message instruction. Table 12.7 the output argument.

Table 12.7 Output Argument

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	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	module detects illegal target node address.
11	0B	module detects illegal file number.
12	0C	module detects illegal target file element offset.
13	0D	module detects illegal target file length.
14	0E	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	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 WORDLENGTH = 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

READY
>
```

CALL 94 – Display Current PRT1 Port Setup

Purpose

Use CALL 94 to display the current PRT1 port configuration on the program port terminal screen. No arguments are PUSHed or POPped.

Syntax

CALL 94

Example

```
>CALL 94

19200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
```

CALL 96 – Clear PRT1 Input/Output Buffers

Purpose

Use CALL 96 to clear port PRT1 input and output buffers. Use the following PUSHes to clear the corresponding buffer:

- PUSH 0 to clear the output buffer
- PUSH 1 to clear the input buffer
- PUSH 2 to clear both buffers

IMPORTANT If port PRT1 is configured for DH485 protocol, this CALL has no effect.

Syntax

PUSH [buffer selection]
CALL 96

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : CALL 96 : REM CLEAR PRT1 OUTPUT BUFFER

READY
>
```

CALL 112 – User LED Control

Purpose

Use CALL 112 to activate or de-activate the user LEDs (LED1 and LED2). Two inputs arguments are required and no output arguments. The first input argument activates or de-activates LED1. The second input argument activates or de-activates LED2. An input argument of one activates the LED. An input argument of zero deactivates the LED. Any other value has no effect on that particular LED.

NOTE When you change to Command mode, the user-defined LEDs remain in their last state until you execute another CALL112.

Syntax

PUSH [LED1 state]
PUSH [LED2 state]
CALL 112

Example

```
>1   REM EXAMPLE PROGRAM
>100 PUSH 1 : REM TURN ON LED1
>110 PUSH 0 : REM TURN OFF LED2
>120 CALL 112 : REM SET THE LEDS

READY
>RUN

READY
>
```

CALL 114 – Transmit DF1 Packet

Purpose

Use CALL 114 to transmit the DF1 data packet. This routine has no input arguments and no output arguments. When CALL 114 is performed, the DF1 data is posted for the DF1 driver to transmit as a single message packet. If half-duplex slave operation is selected, the message packet is transmitted the next time an ENQuiry is received from the DF1 master. If full-duplex operation is selected, the message packet is transmitted immediately.

Use one or more PRINT#, PH0.#, or PH1.# 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.

Caution must be exercised when building DF1 data packets. If an attempt is made 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 module enters Command mode.

If an attempt is made 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 module enters Command mode.

The user's program must wait for one transmission to complete before construction of another data packet may be performed. Use CALL 115 to check the DF1 transmission status to determine when a transmission is complete.

Syntax

CALL 114

Example

```
>1  REM EXAMPLE PROGRAM
>10 CALL 114
>20 END
```

CALL 115 – Check DF1 XMIT Status

Purpose

Use CALL 115 to check the DF1 transmit status. This routine has no input arguments and one output argument. The output argument returns a value that represents the DF1 transmit status. The possible DF1 transmit status values are shown below:

- 0 - No transmit result pending
- 1 - Transmit result pending
- 2 - Transmission successful
- 3 - Transmission failed
- 4 - Enquiry timeout, no transmission
- 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.

IMPORTANT Transmit status value 4 should never be returned if full-duplex mode is selected.

Syntax

CALL 113
POP [DF1 transmit status]

Example

```
>1  REM EXAMPLE PROGRAM
>10 CALL 113
>20 POP X
>30 END
```

CALL 123 – Write to Remote DF1 PLC Data File

Purpose

Use CALL 123 to write up to 64 words of data from the CPU output image file, the CPU M0 file, and/or a string within the module to a remote DF1 node (PLC-2®, -3®, or -5).

The following table lists specific notes when using CALL 123 with the PLC-3 and PLC-5.

Table 12.8 PLC Application Notes

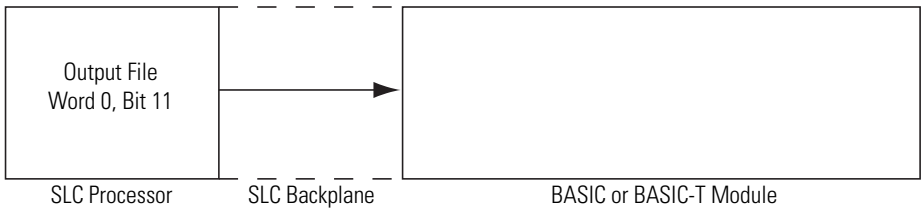
PLC	Notes
-3	For timers and counters, the file number PUSHed (third parameter) is the structure number, limited to a maximum of 255 words.
	Input and output files cannot be accessed with this CALL. Choosing these file types will cause a 2 (bad input parameter) to be POPped.
-5	For timer data, an element is three 16-bit words, stored in the source file in the following order: Control, Preset, and Accumulator.

If an internal string is chosen, the first character (transaction number) is incremented upon a successful write transaction to inform the module that string data was written to the PLC. The value of the transaction number wraps around from 255 to 0.

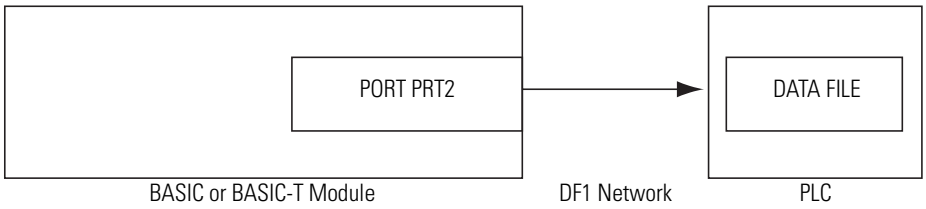
The DF1 port parameters are set up with CALL 108. The DF1 port can operate with either full-duplex or half-duplex slave protocol.

Execute CALL 123 once to set up the data transfer parameters. Input and output image bits (word 0, bit 11) for the slot containing the module, are used to initiate and notify completion of the transfer. The operation is described below:

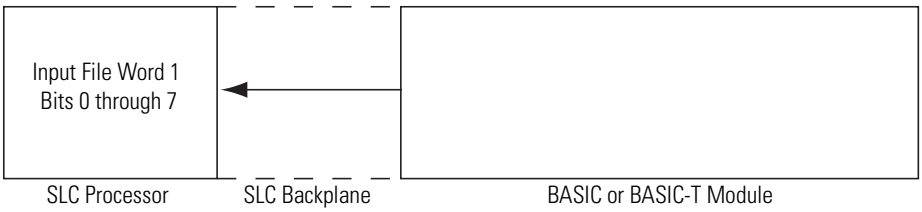
1. The SLC processor builds the data buffer and sets output file word 0, bit 11 to inform the module that valid data is available.



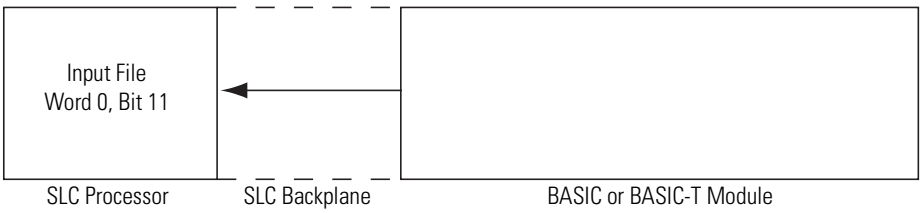
2. The module transfers the data into the destination PLC data file.



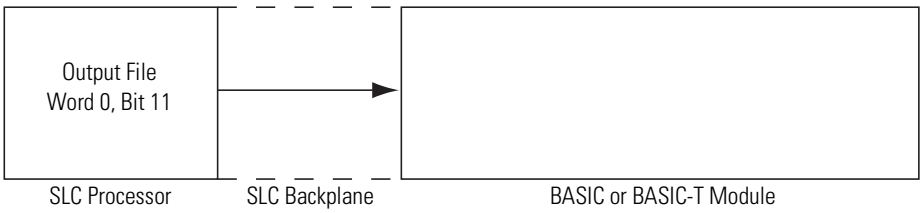
3. The module places the status of the transaction in input file word 1, bits 0-7.



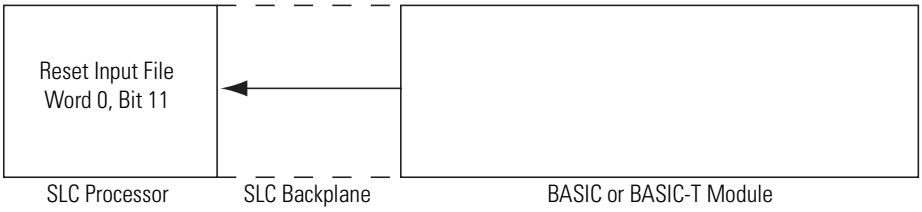
4. The module sets the input file word 0, bit 11 to inform the SLC processor that the data was transmitted, and that the status of the transfer is valid.



5. The SLC retrieves the status and resets output file word 0, bit 11. The data buffer can be reused by the SLC processor.



6. The module resets the input file word 0, bit 11 on the same end of scan cycle in which the output file word 0, bit 11 was reset.



This CALL is active until it is re-executed with different input parameters.

This CALL has ten input arguments and one output argument.

The first input argument is the type of PLC WRITE command issued:

- 0 - Disable the previously executed CALL
- 2 - Common interface file - PLC-2 unprotected WRITE command
- 3 - PLC-3 file - word range WRITE command
- 5 - PLC-5 file - typed WRITE command

The second input argument is the decimal node address of the PLC remote device (0 through 254). If the number is not within this range, the status equals 2 and the write message does not occur.

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. The parameter is PUSHed, but ignored, if the common interface file is chosen in the first parameter.

The fourth input argument is the destination file type on the remote device. This number is ignored if the common interface file is chosen in the first parameter (assumes integer file). If the file type code is not one of these shown below, the status equals 2 and the write message does not take place.

Table 12.9 FileType Written to the Remote Device

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

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.

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.

Table 12.10 Valid Element Length Range

File Type Code	Valid Element Length Range
ASC(N)	1 to 64
ASC(S)	1 to 64
ASC(C)	1 to 21
ASC(T)	1 to 21
ASC(B)	1 to 64
ASC(R)	1 to 21
ASC(I)	1 to 64
ASC(O)	1 to 64
Common Interface File	1 to 64

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, bits 0-7. If the time-out value is not within the range (1 through 255), the POPped status equals 2 and the transfer does not take place.

The eighth input argument is the selection of the source CPU output image file, CPU M0 file, or the internal string:

- 0 - CPU output image file
- 1 - CPU M0 file
- 2 - Internal string

If you chose internal string (2), CALL 29 can be executed to initiate each data transfer without requiring SLC processor interaction. The output file word 0, bit 11 will also initiate a string transaction.

The ninth input argument is the word offset within the CPU file. This offset points to the first word of the data. If you chose the CPU output image file (0), the offset cannot be 0 since this word is reserved for the data transfer handshaking bits. The offset for the internal string is always one. The first character of the string (transaction number at location 0) is incremented on every successful transfer to inform the module that the data was transferred. The value of the transaction number wraps around from 255 to 0.

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 must still be PUSHed.

The output argument is the validation of the CALL. It has the following values:

- 0 - Successful
- 1 - Disabled
- 2 - Bad input parameter
- 3 - DF1 not enabled
- 4 - String too small
- 5 - String is not dimensioned

To disable this CALL, a zero must be PUSHed into the first input parameter. All other parameters are ignored but must still be PUSHed.

Whenever an attempt is made to write to a remote packet, the module places the status of the write into the input word 0, bits 0-7. The possible status codes are shown below. This status is valid when the module sets the input file word 0, bit 11.

Table 12.11 Transaction Status Codes

Code	Indicates
0	Transfer OK.
1	Transmission failed.
2	Enquiry timeout.
3	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.

Table 12.11 Transaction Status Codes

Code	Indicates
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 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 source file]

PUSH [word offset within source file]

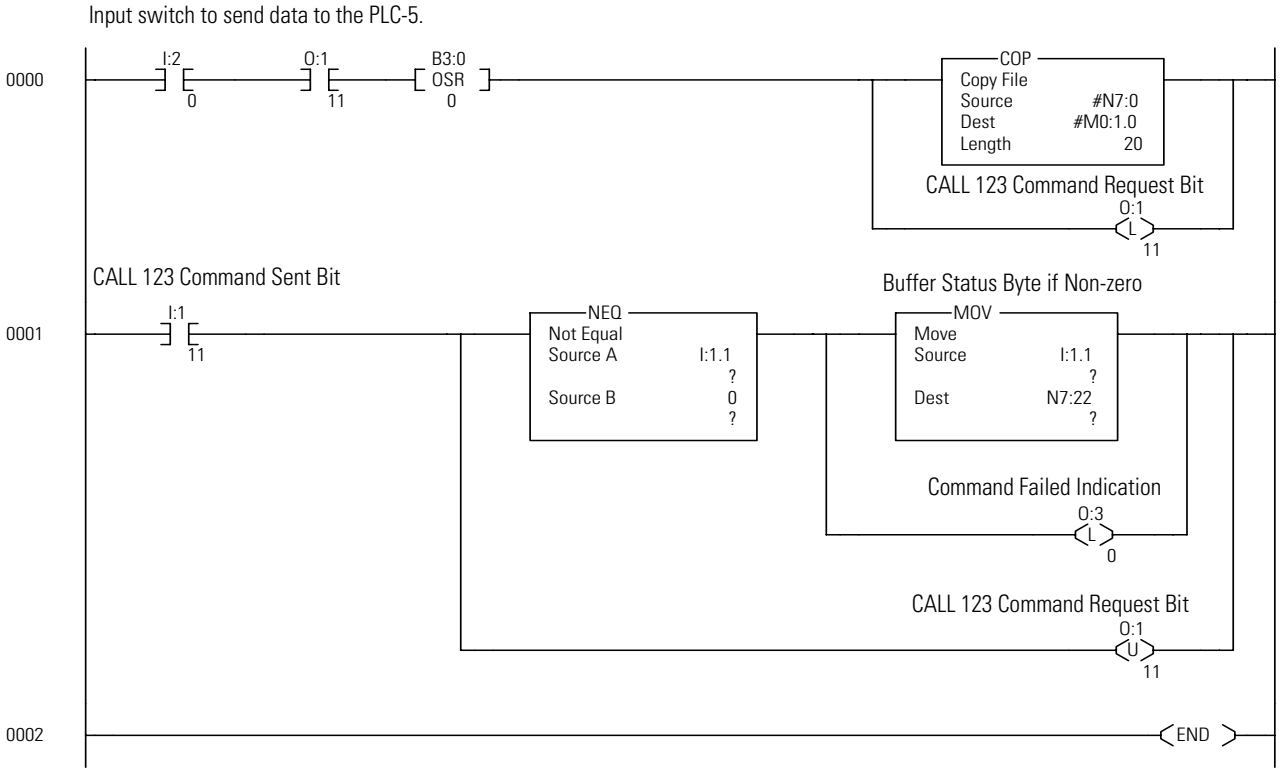
PUSH [string number]

CALL 123

POP [CALL 123 status]

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PLC REMOTE WRITE COMMAND
>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 1 : REM USE M0 FILE
>100 PUSH 0 : REM OFFSET WITHIN M0 FILE
>110 PUSH 0 : REM STRING NUMBER - NOT APPLICABLE FOR THIS
     EXAMPLE
>120 CALL 123
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 123 SETUP"
```



PRINT

Purpose

Use the PRINT statement to direct the 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,

Syntax

PRINT

Example

```
>PRINT 10*10,3*3
 100  9

>PRINT ”1746-BAS”
1746-BAS

>PRINT 5,1E3
 5  1000
```

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 module resumes execution from the point at where it left off. The output buffer of each port is capable of holding 256 characters. See descriptions of CALLs 36, 37, 95, and 96 for more information.

The symbols @ and # can be used to direct the print output to ports PRT1 and PRT2 respectively.

Use the PRINT CR expression to output a carriage return without a line feed.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, CR,
>20 PRINT “B”

READY
>RUN

B

READY
>
```

The A was printed and then overwritten by the B.

Use the PRINT SPC( ) expression to output a specified number of spaces.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, SPC(10), “B”

READY
>RUN

A          B
```

Use the PRINT TAB( ) expression to output a specified number of tab characters.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, TAB(1), “B”

READY
>RUN

A    B
```

Use the PRINT USING(Fx) expression to 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.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT USING(F4), 123.45678

READY
>RUN

1.234 E+2
```

Use the PRINT USING(#.#) expression to output all numeric values in decimal notation according to the format specified by the instruction.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT USING(###.##),
>20 PRINT 4.67890, 123.456
>30 PRINT .0123, .234
>40 PRINT 123.456, 2.1

READY
>RUN

4.67          123.45
0.01          0.23
123.45        2.10

READY
>
```

Use the PRINT USING(0) expression to restore the default print mode if the mode was altered by the PRINT USING(Fx) expression, or by the PRINT USING(#.#) expression.

PH0., PH1.

Purpose

Use the PH0. and PH1. statements to direct the module to output a hexadecimal value to the console device. These statements function the same as the PRINT statement except that the values are printed out in a hexadecimal format. The PH0. statement suppresses two leading zeros if the number printed is less than 255 (0FFH). The PH1. statement always prints out four hexadecimal digits.

The character H is always printed after the number when PH0. or PH1. is used to direct an output. The values printed are always truncated integers. If the number printed is not within the range of valid integer (example: is between 0 and 65535 [0FFFFH] inclusive), the module defaults to the normal mode of print. If this happens no H prints out after the value. Since integers are entered in either decimal or hexadecimal form, the statements PRINT, PH0., and PH1. can be used to perform decimal to hexadecimal and hexadecimal to decimal conversion. All comments that apply to the PRINT statement apply to the PH0. and PH1. statements.

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 module resumes execution from the point at where it left off. The output buffer of each port is capable of holding 256 characters. See descriptions of CALLs 36, 37, 95, and 96 for more information.

Syntax

PH0., PH1.

Example

```
>PH0.2*2
 04H

>PH1.2*2
 0004H

>PH0. 100
 64H

>PH0. 1000
 3E8H

>PH1. 1000
 03E8H

>PH1. 3E8
 3.0 E+8

>PH0. PI
 03H

>
```

ST@

Purpose

Use the ST@ statement to store module floating point numbers to a specified address. The expression [expr] following the ST@ statement specifies the address where the number is 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 specified by [expr].

This statement can be used with CALL 77 to store variables to a protected area of memory. This protected area is not zeroed on powerup or when the RUN command is issued.

Syntax

ST@ [expr]

Example

```
>P. MTOP
 24515
```

```
P. MTOP-10*6
 24455

>PUSH 24455 : CALL 77

>P. MTOP
 24455

>1   REM EXAMPLE PROGRAM
>5   DIM A(10),B(10)
>10  REM *** ARRAY SAVE ***
>20  FOR I = 0 TO 9
>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 9
>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
 25
 26
 27
 28
 29

READY
>PUSH 5FFFH : CALL 77

>P. MTOP
 24575
```

Input Functions

This chapter describes and illustrates commands that allow the BASIC or BASIC-T module to read input data from its external ports within the BASIC program or from the command line. Table 13.1 lists the corresponding mnemonics.

Table 13.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Transfer data from PRT1 or PRT2 to the SLC I/O or M files.	CALL 22	13-2
Transfer data from a remote DH485 data file to the SLC processor.	CALL 27	13-8
Handle all errors that are not handled by the ONERR statement.	CALL 29	13-13
Get a numeric input character from port PRT2.	CALL 35	13-15
Transfer the CPU output image buffer to the module input buffer.	CALL 53	13-17
Transfer the CPU M0 file to the module input buffer.	CALL 56	13-18
Transfer the DH485 interface file to the module input buffer.	CALL 84	13-19
Read remote DH485 data file to the BASIC input buffer.	CALL 90	13-20
Read remote DH485 interface file to the module input buffer.	CALL 92	13-23
Get the DF1 packet length.	CALL 117	13-25
Allow unsolicited writes from a remote SLC or PLC node.	CALL 118	13-26
Read a PLC data file.	CALL 122	13-30
Read the console input device.	GET	13-38
Read the console input device connected to PRT2.	GET#	13-38
Read the console input device connected to PRT1.	GET@	13-38
Read a line of characters from the program port buffer.	INPL	13-39
Read a line of characters from the port PRT2 buffer.	INPL#	13-39
Read a line of characters from the port PRT1 buffer.	INPL@	13-39
Read a string of characters from the program port buffer.	INPS	13-40
Read a string of characters from the port PRT2 buffer.	INPS#	13-40
Read a string of characters from the port PRT1 buffer.	INPS@	13-40
Input a string or variable.	INPUT	13-40
Input a string or variable from port PRT2.	INPUT#	13-40
Input a string or variable from port PRT1.	INPUT@	13-40
Load a variable.	LD@	13-43
READ data in the data statement.	READ	13-45

CALL 22 – Transfer Data from Port 1 or 2 to the CPU Files

Purpose

Use CALL 22 to transfer data from the module serial ports directly to the CPU input data file, CPU M1 file and/or an internal string within the module. During data transfer, data is automatically transferred in 8-bit blocks from the input buffer of the selected port to the selected SLC processor buffer and/or BASIC internal string for storage. The transfer occurs when the specified number of characters are detected in the input buffer of the port or the user-defined delimiter is received in the port. The data is stored 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 an odd number of bytes are to be transferred, the unused byte contains a zero.

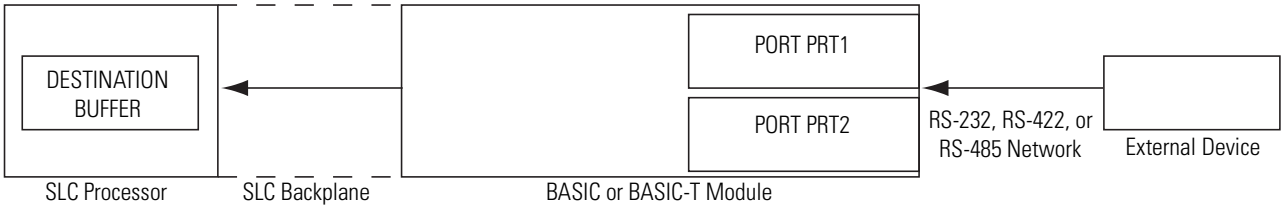
The byte swap selection (low byte first, then high byte, or high byte first, then low byte) of the last CALL 22 or CALL 23 executed determines the data packing method for all ports enabled by CALL 22.

The low byte of the first word of the destination file contains the character count (byte count) of the data being transferred. If a delimiter is found, the byte count is expanded to include the first occurrence of the delimiter. The second word of the destination file contains the first two characters of data.

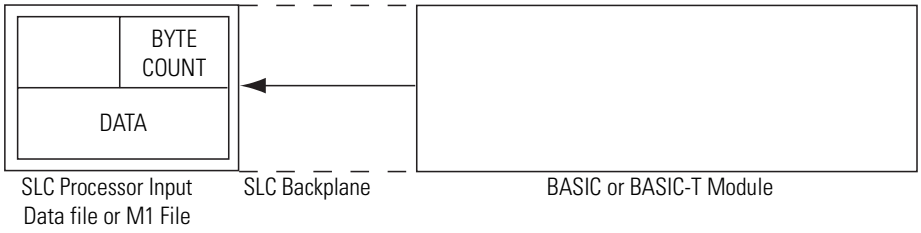
If an internal string is chosen, the first character of the string contains the byte count. The second character of the internal string is a transaction number and is incremented to inform the module that new data is in the string. The value of this character wraps around from 255 to 0. The third character of the string contains the first data character.

Execute CALL 22 to set up the data transfer parameters. After the CALL is executed, the module gets data from the port and transfers it to the destination file. Input and output image file bits (word 0, bits 8 and 9) for the slot containing the module, are used to initiate and notify completion of the transfer. The operation is described below:

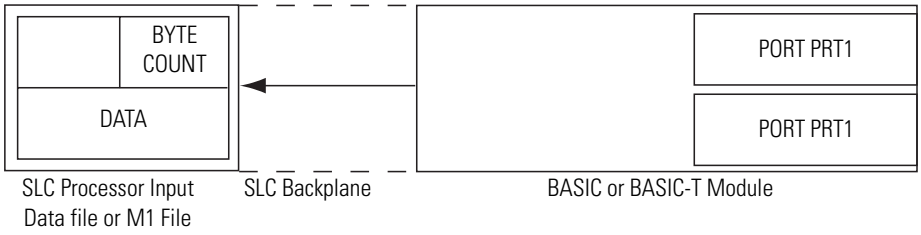
1. When data is available from the port, the module automatically transfers the data into the destination buffer. This same port is checked for data at the end of each line of BASIC code.



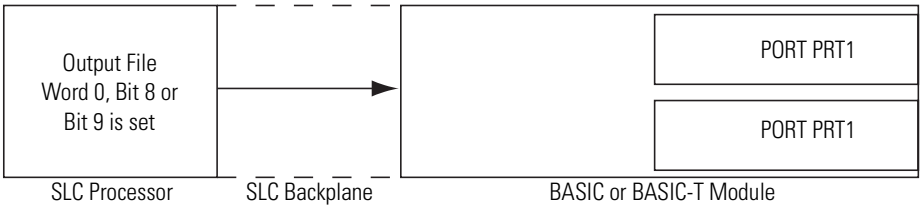
2. The module places the byte count of the valid data into the lower byte of the first available word of the destination buffer. The upper byte of the first available word is not used.



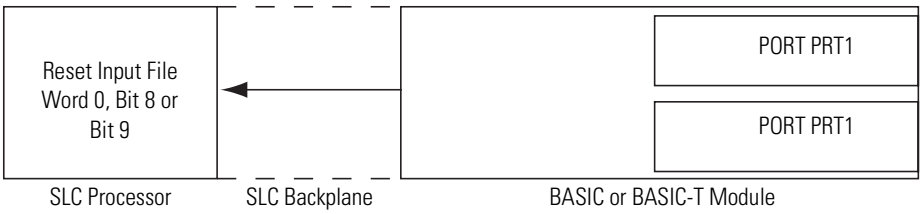
3. The module sets the input file word 0, bit 8 or bit 9 to inform the SLC processor that valid data is available. Bit 8 indicates that data is available from port 1 and bit 9 indicates that data is available from port 2.



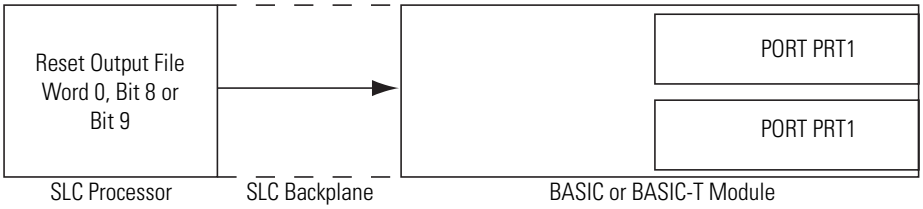
4. The ladder logic program of the SLC processor gets the data from the file. The ladder logic program sets the output file word 0, bit 8 or bit 9 to inform the module that data was received.



5. The module resets the input file word 0, bit 8 or bit 9 on the same end of scan cycle in which the output file word 0, bit 8 or bit 9 was set.



6. The ladder logic program of the SLC resets output file word 0, bit 8 or bit 9. The module can begin loading the destination buffer with the next packet as data arrives from the port.



Data transfers continue until the CALL for the port is re-executed with different input parameters. If this occurs, the previous CALL 22 for the port is automatically disabled and the new CALL 22 takes effect. Multiple CALL 22s for the same port are not executed in parallel. However, port 1 and port 2 can be activated simultaneously by issuing separate CALL 22s.

This CALL has seven input arguments and one output argument.

The first input argument is the source port number (1 or 2) of the module. A zero disables all previously active CALL 22 commands.

- 0 - Disable CALL 22 for all active ports enabled by earlier CALL 22s.
- 1 - Port 1 is source.
- 2 - Port 2 is source.

The second input argument is the maximum number of 8-bit characters to be copied from the BASIC serial port to the destination file. The maximum number of characters is selected by the fourth input argument:

- CPU input image file: maximum characters - 10 (5 words)
- CPU M1 file: maximum characters - 126 (63 words)
- Internal string: maximum characters - string size-3 (The first character is the byte count value; the second character is the incremented transaction number; and the last character is the terminating character. The maximum number of characters for an internal string is 254.)

If less than the maximum is acquired when a delimiter character is received, the packet including the delimiter is sent to the CPU file. The SLC processor determines the amount of valid data transferred into the destination file from the byte count placed into the lower byte of the first word of the file.

If the data received exceeds the string length or CPU file size, the remaining data is truncated.

The third input argument is the decimal value of the ASCII character delimiter. Any valid ASCII character can be chosen. If no delimiter is desired, enter a NULL value (0 decimal). The data will be transferred to the destination buffer when the delimiter is received from the selected port regardless of the number of characters received.

The fourth input argument is the selection of the destination CPU input image file with or without the internal string, the CPU M1 file with or without the internal string, or the internal string alone:

- 0 - CPU input image file
- 1 - CPU M1 file
- 2 - CPU input image file and internal string
- 3 - CPU M1 file and internal string
- 4 - Internal string only

When transferring data to the internal string of the 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.

The fifth input argument is the word offset within the destination CPU file. If the CPU input data file is chosen, this offset must not be 0 or 1, since word 0 and 1 are reserved. Zero is reserved for data transfer handshaking bits and word 1 is reserved for the transaction status. A 0 or 1 will cause a CALL status of 2 to be POPped. This offset points to the buffer location of the byte count. If the M1 file is chosen, the offset can be zero. If the internal string is chosen, data placement always begins with the third character of the string. (The first character contains the byte count and the second character contains the transaction number.) Therefore, the offset value has no effect on string data placement, only on input image file and M1 file data placement.

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 must still be PUSHed.

The seventh input argument is the byte swap selection. It has the following values:

- 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 22 or CALL 23 executed determines the byte swap option for all active CALL 22 commands previously executed.

The output argument is the status of the CALL. It has the following 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

If data is being received into the serial port faster than the SLC 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 destination file and/or string]

PUSH [word offset within the destination file]

PUSH [string number]

PUSH [byte swap selection]

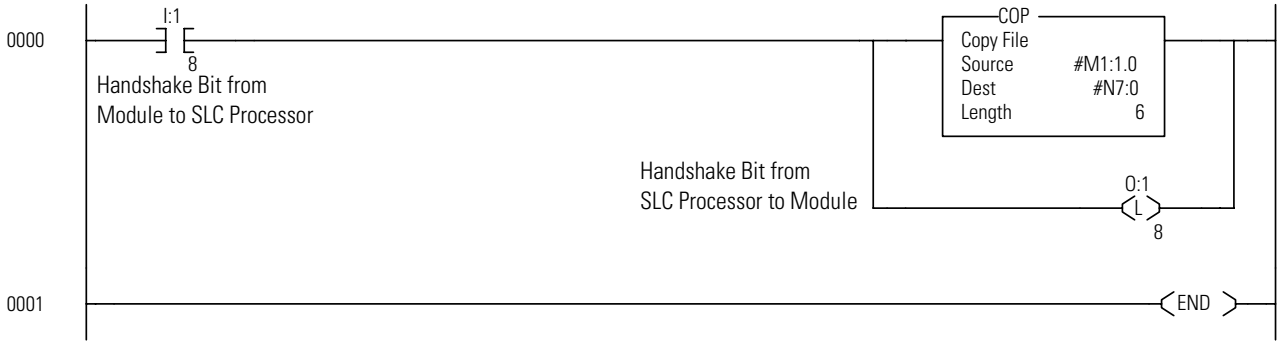
CALL 22

POP [CALL 22 status]

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE CALL 22 INTERRUPTS
>20  PUSH 1 : REM PRT1 ACTIVE FOR CALL 22
>30  PUSH 10 : REM RECEIVING 10 BYTES OF DATA MAXIMUM
>40  PUSH 13 : REM <CR> USED AS TERMINATION CHARACTER
     (13 DECIMAL)
>50  PUSH 1 : REM SEND DATA TO M1 FILE
>60  PUSH 0 : REM OFFSET INTO M1 FILE
>70  PUSH 0 : REM STRING NUMBER - NOT USED
>80  PUSH 1 : REM BYTE SWAPPING ENABLED
>90  CALL 22
>100 POP S : REM STATUS OF CALL 22 SETUP
>110 IF (S<>0) THEN PRINT "UNSUCCESSFUL CALL 22 SETUP"
>120 END
```

Below is a sample ladder logic program for CALL 22. The module is located in slot 1 of the SLC chassis. At rung 0000, copy data from the M1 file when the handshake bit (I:1.0/8) is set by the module. The SLC sets handshake bit (O:1.0/8) once the data has been copied out of the M1 file. This informs the module to turn off (I:1.0/8). The first word of the M1 file contains the byte count and this word is not included in the data byte count. A maximum of 10 bytes of data is expected in this example.



CALL 27 – Read Remote DH485 SLC Data File

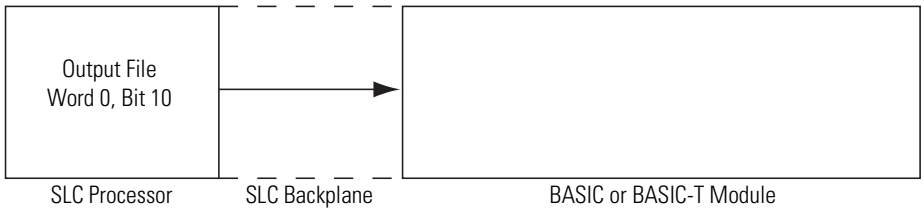
Purpose

Use CALL 27 to read up to 64 words of data from a remote DH485 node data file to the local CPU input image file, the CPU M1 file, and/or a string within the module.

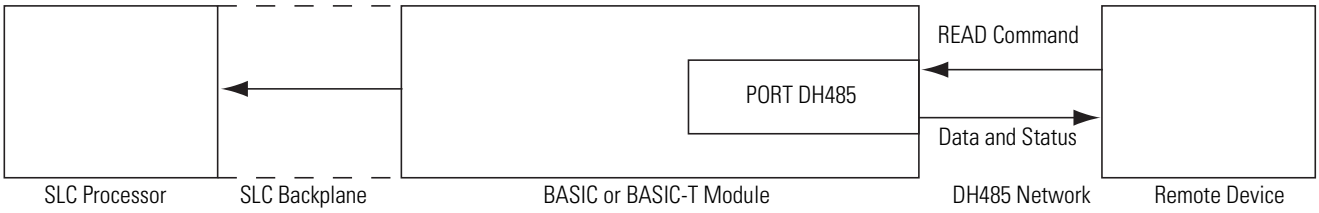
If an internal string is chosen, the first character is incremented to inform the module that new data is in the string. The value of this character wraps around from 255 to 0.

Execute CALL 27 once to set up the data transfer parameters. Input and output image bits (word 0, bit 10) for the slot containing the module, are used to initiate and notify completion of the transfer. The module sends the READ command configured in CALL 27 to the designated remote DH485 device on the network. The operation is described below:

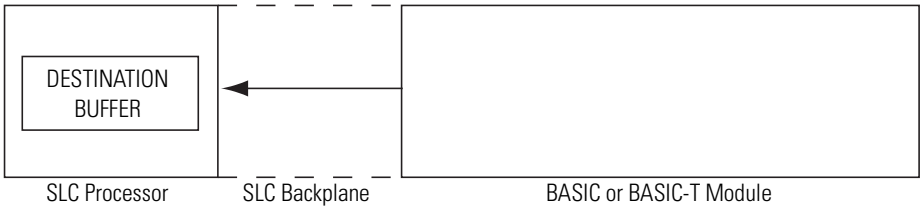
1. The local SLC processor sets the output file word 0, bit 10 to inform the module that the READ command configured in CALL 27 should be executed.



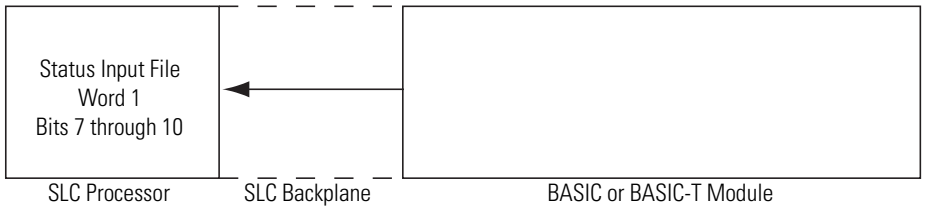
2. The module automatically issues the appropriate READ command to the remote device on the DH485 network. The data and status are sent back to the local SLC processor.



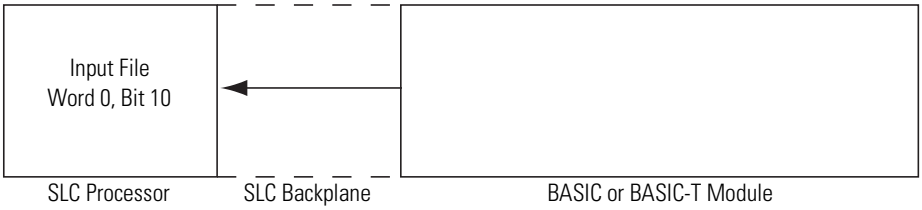
3. When data is available, the module transfers the data into the local SLC destination buffer.



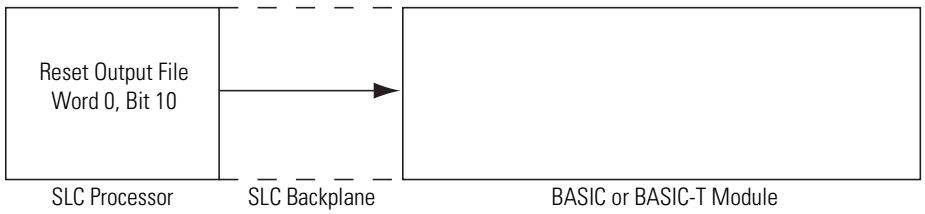
4. The module places the status in the input file word 1, bits 0-7.



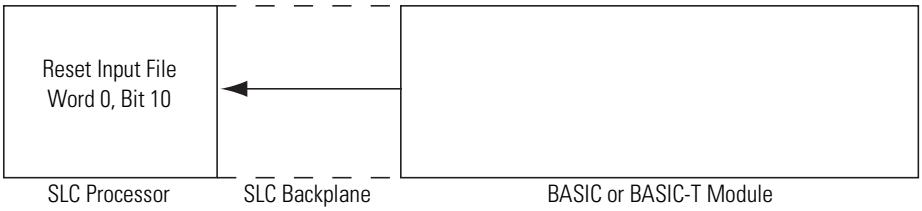
5. The module sets the input file word 0, bit 10 to inform the SLC processor that valid data is available.



6. The local SLC retrieves the data and status from the buffer and then resets output file word 0, bit 10 to inform the module that data was received.



7. The module resets the input file word 0, bit 10 on the same end of scan cycle in which the output file word 0, bit 10 was reset.



The SLC processor must not set, then reset, the output file word 0, bit 10 on the same ladder logic scan cycle. If this occurs, the module may miss the bit being set.

This CALL is active until it is re-executed with different input parameters.

This CALL has ten input arguments and one output argument.

The first input argument is the type of SLC READ command issued:

- 0 - Disable the previously executed CALL 27
- 1 - Common Interface File Read
- 2 - SLC Typed Read

The second input argument is the node address of the SLC remote device (0 through 31). If the number is not within this range, the status equals 2 and the read message does not occur.

The third input argument is the file number on the SLC 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 (CIF) is chosen in the first parameter. The CIF is always file 9.

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.

Table 13.2 FileType to be Read from Remote Device

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

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.)

The sixth input argument is the number of words to be transferred. 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.

Table 13.3 Valid Length Range

File Type Code	Valid Length Range
ASC(N)	1 to 64
ASC(C)	1 to 21
ASC(T)	1 to 21
ASC(B)	1 to 64
ASC(R)	1 to 21
Common Interface File	1 to 64

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, bits 0 through 7. If the time-out value is not within the range (1 through 255), the POPped status equals 2 and the transfer does not take place.

The eighth input argument is the selection of the destination file and/or string:

- 0 - CPU input image file
- 1 - CPU M1 file
- 2 - Internal string
- 3 - CPU input image file and internal string
- 4 - CPU M1 file and internal string

If you chose internal string (2), CALL 29 can be executed to initiate each data transfer without requiring SLC processor interaction. The output file word 0, bit 10 will also initiate a string transaction.

The ninth input argument is the word offset within the destination CPU file. This offset points to the first word transferred. The offset for the internal string is always 1 (the transaction number at location 0) followed by the data. The transaction number is incremented upon every data transfer and wraps around from 255 to 0.

If the CPU input image file is chosen, this offset must not be 0 or 1. Zero is reserved for data transfer handshaking bits and word 1 is reserved for the transaction status. A 0 or 1 causes an error to be POPped (2 - bad input parameter) and the CALL is not executed.

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 must still be PUSHed.

The output argument is the status of the CALL. It has the following values:

- 0 - Successful
- 1 - Disabled
- 2 - Bad input parameter
- 3 - Port DH485 not enabled (DF1 enabled)
- 4 - String is too small
- 5 - String is not dimensioned

To disable this CALL, a 0 must be PUSHed into the first input parameter. All other parameters are ignored but must still be PUSHed.

Whenever an attempt is made to read a remote packet, the status of the read is placed into input word 1, bits 0 through 7. These values have the same definition as the values POPped in CALL 92. The status becomes valid when the module sets the input file word 0, bit 10.

Syntax

PUSH [type of 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]

PUSH [word offset within destination file]

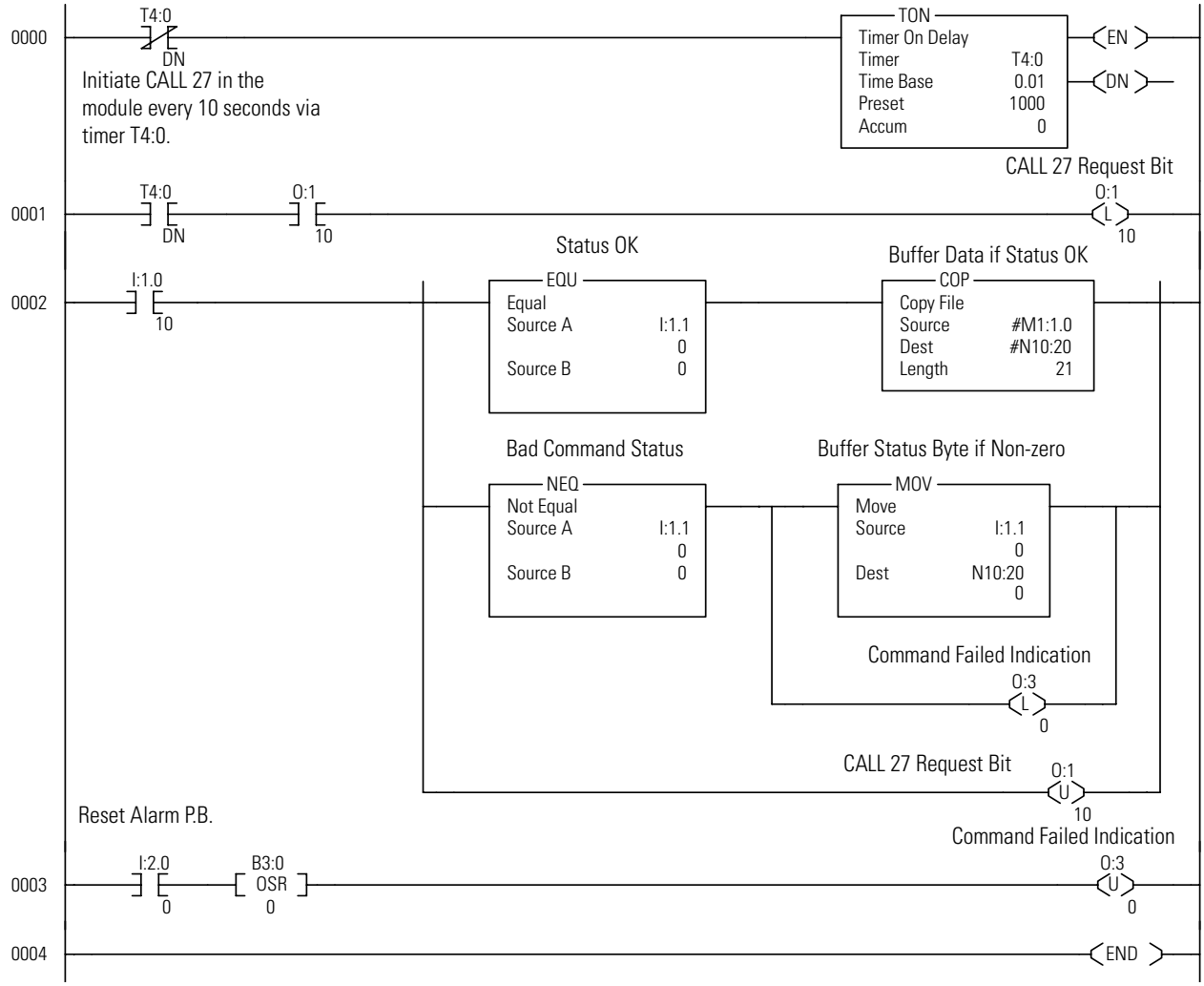
PUSH [string number]

CALL 27

POP [CALL 27 status]

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE REMOTE DH485 READ COMMAND INTERRUPT
>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 1 : REM DESTINATION FILE TO PUT DATA (M1 FILE)
>100 PUSH 0 : REM WORD OFFSET INTO DESTINATION FILE
>110 PUSH 0 : REM STRING NUMBER - NOT AVAILABLE FOR THIS
     EXAMPLE
>120 CALL 27
>130 POP S
>140 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 27 SETUP”
```



CALL 29 – Read/Write to a PLC/SLC from the Module Internal String

Purpose

Use CALL 29, in conjunction with CALLs 122 or 123, to communicate between remote PLCs and the module internal string without local SLC processor interaction. You can also use CALL 29, in conjunction with CALLs 27 or 28, to communicate between remote SLCs and the module internal string without local SLC processor interaction. CALLs 27, 28, 122, or 123 must be executed within the BASIC program before CALL 29.

CALL 29 is active when the internal string is the only choice in CALLs 27, 28, 122, or 123. In this situation, it is not practical to use the SLC input and output image files to begin the transfer and to pass the status. The SLC processor does not need to be involved. If an SLC file is chosen instead, the local SLC processor controls the transfer with the input and output image bits. In this instance, a status of 255 is returned when CALL 29 is attempted.

One argument is PUSHed and one argument is POPped. The input argument is the CALL to be activated (CALL 27, 28, 122, or 123).

When CALL 29 is executed, the transfer is attempted. If the selected CALL (27, 28, 122, or 123) is not executed before CALL 29, a 1 is returned in the status POPped. When CALL 29 is executed successfully, the value in the first character of the string (transaction number) is incremented to designate that the transfer occurred. The range of this character is 0 through 255.

After CALL 29 is executed, one word is POPped. This word is the status of the transaction:

- 0 – Successful completion
- 1 – The chosen CALL (27, 28, 122, or 123) is not active
- 255 – SLC buffer is chosen for CALL 27, 28, 122, or 123 and CALL 29 is ignored
- All other codes are identical to CALL 90/92

Syntax

PUSH [27, 28, 122, or 123 for the CALL you want activated]

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 at node 3, to the internal string $(1) of the module.

```
>1   REM EXAMPLE PROGRAM
>10  REM EXECUTE DF1 PLC REMOTE READ FROM INTERNAL
>20  REM STRING WITH NO SLC INTERVENTION
>21  REM SET UP CALL 122
>25  PUSH 5, 3, 10, ASC(N), 0, 10, 10, 1, 1, 1: CALL 122: 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 “SLC FILE CHOSEN FOR CALL 122”
>80  IF (S=0) THEN PRINT “SUCCESSFUL TRANSFER”
>90  IF (S<>0) THEN PRINT “UNSUCCESSFUL TRANSFER”
>100 END
```

CALL 29 replaces the handshaking bit function in CALLs 27, 28, 122, and 123 when using an SLC file or module string.

CALL 35 – Get Numeric Input Character from PRT2

Purpose

Use CALL 35 to retrieve the current character in the 256 character input buffer of port PRT2. It returns the decimal representation of the characters received as its output argument. Port PRT2 receives data transmitted by your device and stores it in this buffer. If there is no character, the output argument is a 0 (null). If there is a character, the output argument is the ASCII value of that character. There is no input argument for this routine.

Syntax

CALL 35
POP [ASCII value of character]

Examples

Example 1

```
>1  REM EXAMPLE PROGRAM
>10 CALL 35
>20 POP X
>30 IF X=0 THEN GOTO 10
>40 PRINT CHR(X)

READY
>RUN

A
READY
>
```

The above example assumes the PRT2 input buffer contains the ASCII character A.

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.

IMPORTANT Purge the buffer before storing data to ensure data validity.

Example 2

```
>1   REM EXAMPLE PROGRAM
>10  REM PERIPHERAL PORT INPUT USING CALL 35
>20  STRING 200,20
>30  DIM D(254)
>40  CALL 35 : POP X
>50  IF X <>2 GOTO 40
>55  REM WAIT FOR DEVICE TO SEND START OF TEXT
>60  REM
>70  DO
>80  I=I+1
>90  CALL 35 : POP D(I): REM STORE DATA IN ARRAY
>100 UNTIL D(I)=3 : REM WAIT FOR DEVICE TO SEND END OF TEXT
>120 REM
>130 REM FORMAT AND PRINT DATA TYPES
>140 PRINT “RAW DATA=”
>150 FOR J=1 TO I : PRINT D(J),: NEXT J
>155 REM PRINT RAW DECIMAL DATA
>160 PRINT: PRINT: PRINT
>170 PRINT “ASCII DATA=”
>180 FOR J=1 TO I : PRINT CHR(D(J)),:NEXT J
>185 REM PRINT ASCII DATA
>190 PRINT: PRINT: PRINT
>200 PRINT “$(1)=”
>210 FOR J=1 TO I: ASC($(1),J)=D(J): NEXT J
>215 REM STORE DATA IN STRING
>220 PRINT $(1)
>230 PRINT: PRINT: PRINT
>240 I=0
>250 REM
>260 GOTO 40

READY
>RUN

RAW DATA=
 65 66 67 68 69 70 71 49 50 51 52 53 54 55 56 57 3

ASCII DATA=
 ABCDEFG123456789

$(1)=
 ABCDEFG123456789
```

CALL 53 – Transfer CPU Output Image to BASIC Input Buffer

Purpose

Use CALL 53 to transfer words 0 to 7 of the CPU output image table to words 200 to 207 of the module input buffer. This routine has no input arguments and one output argument. The output argument is the status of the Logic Processor. It can have one of the following values:

- 0 – Logic processor is in the Run mode
- 1 – Logic processor is not in the Run mode

Word integrity is guaranteed during this transfer. File integrity is not. Handshaking bits can be used in your application program to provide file integrity.

All data transferred to the module from the SLC 500 processor must be routed through the module input buffer. Table 13.4 lists the definition of the addresses in the module input buffer.

Table 13.4 Module Input Buffer Addresses

Address	Definition
0 to 39	Data transferred from the DH485 common interface file.
40 to 99	Reserved
100 to 163	Data transferred from the SLC 500 CPU module M0 file.
164 to 199	Reserved
200 to 207	Data transferred from the SLC 500 CPU output image table.

Syntax

CALL 53
POP [processor status]

Example

```
>1 REM EXAMPLE PROGRAM
>30 CALL 53 : REM XFER CPU OUTPUT IMAGE TO BASIC INPUT BUFFER
>40 POP X : REM LOGIC PROCESSOR STATUS
>50 IF (X<>0) THEN PRINT “PROCESSOR NOT IN RUN MODE”

READY
>RUN
```

CALL 56 – Transfer CPU M0 File to BASIC Input Buffer

Purpose

Use CALL 56 to transfer up to 64 words starting at word 0 of the CPU M0 file to the module input buffer starting at word 100. This routine has one input argument and one output argument. The input argument is the number of words to be transferred (0 to 64). If the number is not within the range 0 to 64, no transfer occurs, and the output argument sets to 10. If the input argument is valid number, the output argument is the status of the Logic Processor. It can have one of the following values:

- 0 – Successful Transfer, Logic Processor in Run mode
- 1 – Successful Transfer, Logic Processor in Program mode
- 2 – Successful Transfer, Logic Processor in Test mode
- 10 – Illegal length specified
- 11 – Logic Processor does not support this capability

Word integrity is guaranteed during this transfer. File integrity is not. Handshaking bits can be used in your application program to provide file integrity.

Syntax

PUSH [number of words to be transferred]
CALL 56
POP [processor status]

Example

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 64 :  REM TRANSFER 64 WORDS
>40 CALL 56 :  REM TRANSFER M0 TO BASIC INPUT BUFFER
>50 POP X :  REM LOGIC PROCESSOR STATUS IS IN X
>60 IF (X=10) PRINT “ILLEGAL INPUT ARGUMENT”
>70 IF (X<>0).AND.(X<>10) THEN  PRINT “PROCESSOR NOT IN RUN
    MODE”

READY
>RUN
```

CALL 84 – Transfer DH485 Interface File to BASIC Input Buffer

Purpose

Use CALL 84 to transfer up to 40 words starting at the designated offset of the DH485 Common Interface File to the module input buffer starting at the same designated offset from word 0. This routine has two input arguments and one output argument. The first input argument is the starting offset in the DH485 Common Interface File and the module input buffer (0 to 39). If the number is not within the range 0 to 39, the output argument equals 1, and the transfer does not take place. The second input argument is the length in words to be transferred (1 to 40). If the number of words is not within the range 1 to 40, the output argument equals 2, and the transfer does not take place.

- 0 – Successful transfer
- 1 – Illegal starting offset
- 2 – Illegal length

Word integrity is guaranteed during this transfer. File integrity is not. Handshaking bits can be used in your application program to provide file integrity.

Syntax

PUSH [starting word offset in DH485 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 : REM
    PRINT ERROR

READY
>RUN

READY
>
```

CALL 90 – Read Remote DH485 Data File to BASIC Input Buffer

Purpose

Use CALL 90 to read up to 40 words from the designated node address, file number, file type, and element offset of a remote DH485 data file to the module input buffer starting at word 0. This routine has six input arguments and one output argument.

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.

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 read message does not take place.

The third input argument is the file type read from the remote device. Valid file type codes are ASC(N), ASC(S), ASC(C), ASC(T), ASC(B), and ASC(R). If the file type is not one of these valid types, then the output argument equals 241, and the read message does not take place.

Table 13.5 FileType to be Read from Remote Device

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

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 will be twice of what is expected. For example, if an offset of 3 was PUSHed, the data will be written to the remote DH485 data file beginning at element 6.

The fifth input argument is the number of elements to be transferred. If the number is not within the valid length range specified in Table 13.6, then the output argument equals 13, and the transfer does not take place.

Table 13.6 Valid Length Range

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

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 module input buffers starting at word 0 and filling as many words as specified by the element length of the message.

The output argument specifies the status of the message instruction. Upon return from the CALL, the output argument has the following definition.

Table 13.7 Output Argument

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	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	module detects illegal target node address.
11	0B	module detects illegal file number.
12	0C	module detects illegal target file element offset.
13	0D	module detects illegal target file length.
14	0E	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.

Table 13.7 Output Argument

Decimal Output	Hexadecimal Output	Description
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	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

```
>1  REM EXAMPLE PROGRAM
>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 TIMEOUT = 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 92 – Read Remote DH485 Common Interface File to BASIC Input Buffer

Purpose

Use CALL 92 to read up to 40 words from the remote DH485 Common Interface File of the designated node address, starting at the designated word offset to the module input buffer starting at word 0. This routine has four input arguments and one output argument.

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.

The second input argument is the starting word 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 will be twice of what is expected. For example, if an offset of 3 was PUSHed, the data will be written to the remote DH485 data file beginning at element 6.

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.

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 module input buffer starting at word 0 and filling as many words as specified by the word length of the message.

Upon return from the CALL, the output argument specifies the status of the message instruction. Table 13.8 defines the output arguments.

Table 13.8 Output Argument

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	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	module detects illegal target node address.
11	0B	module detects illegal file number.
12	0C	module detects illegal target file element offset.
13	0D	module detects illegal target file length.
14	0E	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	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 : REM PRINT
    ERROR

READY
>RUN

READ ERROR CODE IS 5
```

CALL 117 – Get DF1 Packet Length

Purpose

Use CALL 117 to get the length of the DF1 data packet. 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.

IMPORTANT If the receive buffer is found empty, then 0000 is returned to the argument stack.

When CALL 117 is read in a program, the module checks to see if DF1 communications has been enabled through CALL 108. If DF1 communications have not been enabled, an error message is printed to the console device and the module enters Command mode.

After the length of the DF1 packet has been retrieved, it must be used in conjunction with the GET statement to retrieve the data in the received DF1 packet.

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

Purpose

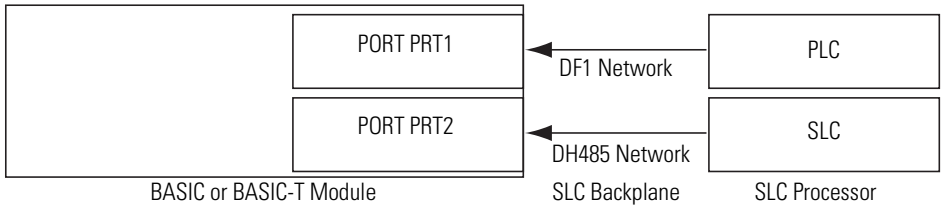
Use CALL 118 to allow the module to receive data packets sent by PLC-2, PLC-3, or PLC-5 message instructions on the DF1 network. This CALL also sets up the module to receive data packets from an SLC node on the DH485 network. Both the DF1 port (PRT2) and the DH485 port cannot be active at the same time. Jumper JW4 on the module is used to select your port configuration.

Any write message instruction sent to the module from these PLC/SLCs cause the data to be placed in an internal string within the CPU M1 file, the CPU input image file and/or the module string, starting at the designated word offset.

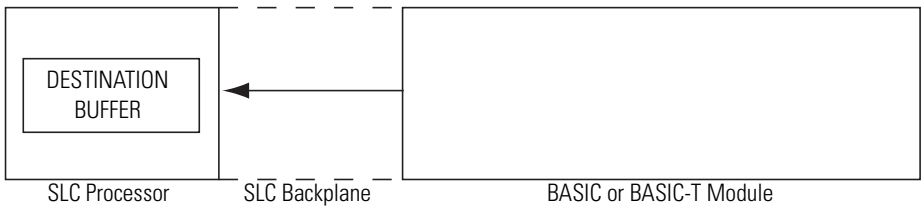
The low byte of the first word of the destination file contains the character count (byte count) of the transferred data. The high byte of this word is unused. If an internal string is chosen, the first character contains the byte count. The second character (transaction number) of the internal string is incremented, upon successful receipt of a packet, to inform the 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 module checks the port at the end of each line of BASIC code. The module gets new data from the PLC or SLC and transfers it as described below:

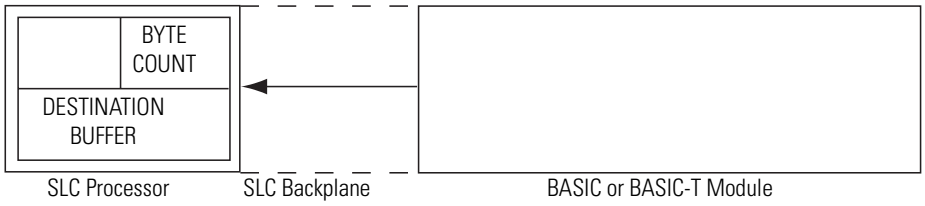
1. The module receives packets initiated from the PLC/SLC configured in the CALL through either ports PRT2 or DH485. Both PRT2 and DH485 ports cannot be active at the same time. Jumper JW4 is used to make this selection.



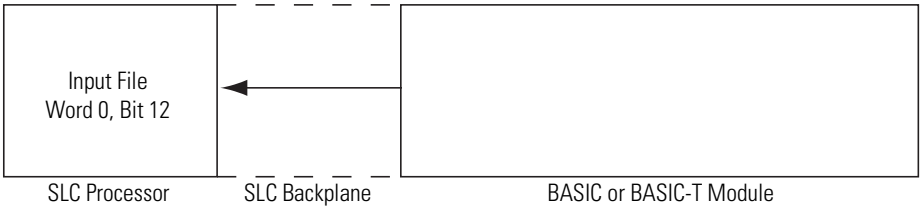
2. The module transfers the data into the local SLC destination buffer.



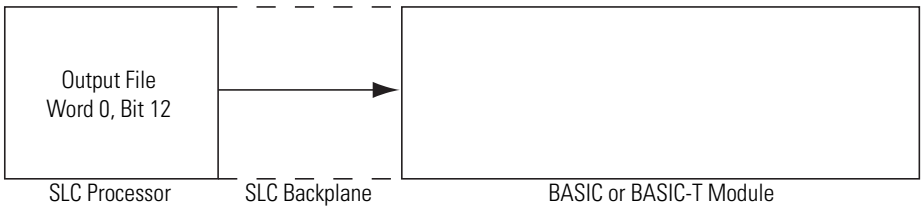
3. The module places the byte count into the lower byte of the first available word of the destination buffer.



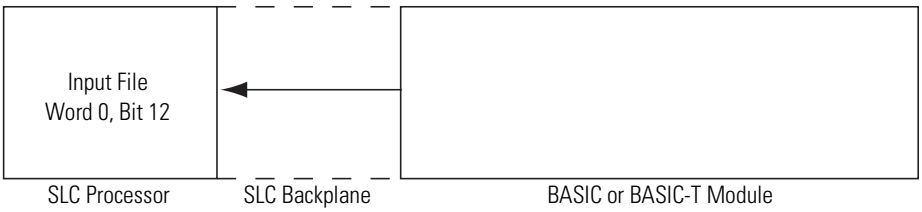
4. The module sets the input file word 0, bit 12 to inform the SLC processor that valid data is available.



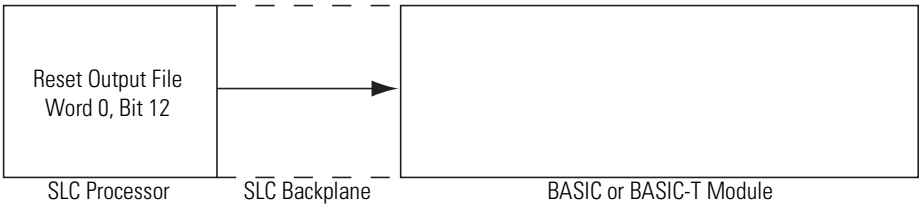
5. The SLC processor retrieves the data from the buffer and sets the output file word 0, bit 12 to inform the module that data was received.



6. The module resets the input file word 0, bit 12 on the same end of scan cycle in which the output file word 0, bit 12 was reset.



7. The SLC resets output file word 0, bit 12. The module can begin loading the destination buffer with the next packet when it is received.



The SLC processor must not set, then reset, the output file word 0, bit 12 on the same ladder logic scan cycle. If this occurs, the module may miss the bit being set.

This CALL is active until it is re-executed with different input parameters. If this occurs, the previous CALL 118 is automatically disabled and the new CALL 118 takes effect. Multiple CALL 118s are not executed in parallel.

This CALL has five input arguments and one output argument.

The first input argument enables or disables the CALL:

- 0 – Disable the previously executed CALL 118
- 1 – Enable the CALL. The following 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.

The second input argument is the selection of the destination CPU input image file with or without the internal string, the CPU M1 file with or without the internal string, or the internal string alone:

- 0 – CPU input image file
- 1 – CPU M1 file
- 2 – Internal string
- 3 – CPU input image file and internal string
- 4 – CPU M1 file and internal string

If the internal string (2) is chosen, the input/output image data handshaking bits (word 0, bit 12) are not used to indicate that data was received by the 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 will be lost.

The third input argument is the word offset within the destination CPU file. This offset points to the first word which contains the byte count of the valid data that is transferred. The offset for the internal string is always 2. The byte count is placed at character location 0, and location 1 is a transaction number that is incremented upon every successful packet completion.

If the DH485 port is used for data transfer, an offset of greater than 40 hex (64 decimal) words should not be used. Unsolicited write packets of greater than 64 causes a write to the DH485 program port buffer, leading to improper operation. The size of the data packet can be up to the maximum for the input file selected.

If the CPU input image file is chosen as the destination, this offset must not be 0 or 1. Zero is a reserved word for the handshaking bits. Word 1 is reserved for the PLC transaction number used with CALLs 27, 28, 122, and 123. A 0 or 1 causes an error to be POPped (2 = bad input parameter) and the CALL is not executed.

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 must still be PUSHed.

The fifth input argument is the maximum word length allowed for the data packet. Any packets received by the module of greater size are rejected. Entering 0 causes the 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.

The output argument is the status of the CALL. It has the following values:

- 0 – Successful
- 1 – Disabled
- 2 – Bad input parameter
- 3 – Selected DH485/DF1 port not enabled
- 4 – String too small
- 5 – String is not dimensioned

Syntax

PUSH [CALL enable/disable]

PUSH [selection of destination file and/or string]

PUSH [word offset in destination file]

PUSH [string number]

PUSH [maximum word length]

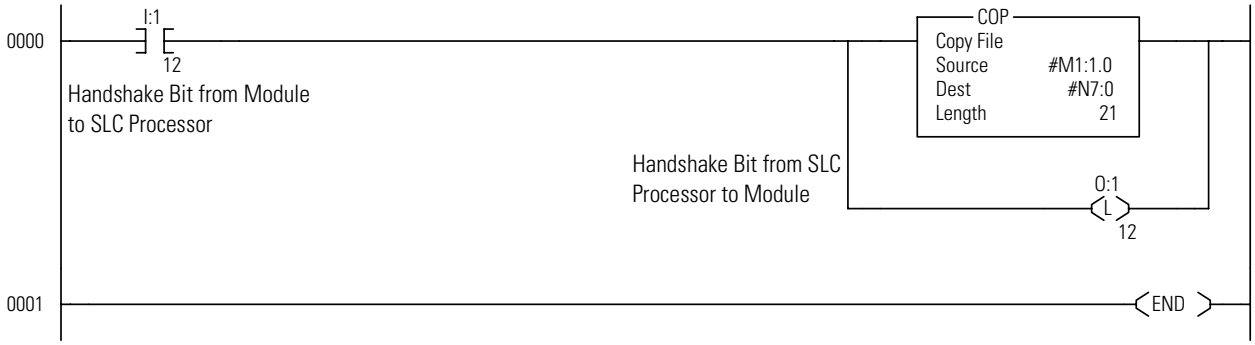
CALL 118

POP [CALL 118 status]

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE PLC/SLC UNSOLICITED WRITE INTERRUPT
>20 PUSH 1 : REM ENABLE THE CALL
>30 PUSH 1 : REM DESTINATION SLC M1 FILE
>40 PUSH 0 : REM WORD OFFSET INTO M1 FILE
>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”
>100 GOTO 100 : REM CALL 118 stays active while BASIC program is
running
```

Below is a sample ladder logic program for CALL 118. The module is located in slot 1 of the SLC rack. Rung 0000 copies data from the M1 file when the handshake bit (I:1.0/12) is set by the module. Rung 2:0 sets the handshake bit (O:1.0/12) once the data has been copied out of the M1 file. This informs the module to turn off I:1.0/12. The first word is the byte count. A maximum of 20 words of data is expected.



CALL 122 – Read Remote DF1 PLC Data File

Purpose

Use CALL 122 to read up to 64 words of data from a remote DF1 node (PLC-2, -3, or -5) to the CPU input image file, the CPU M1 file, and/or a string within the module.

The following table lists specific notes when using CALL 122 with the PLC-3 and PLC-5.

Table 13.9 PLC Application Notes

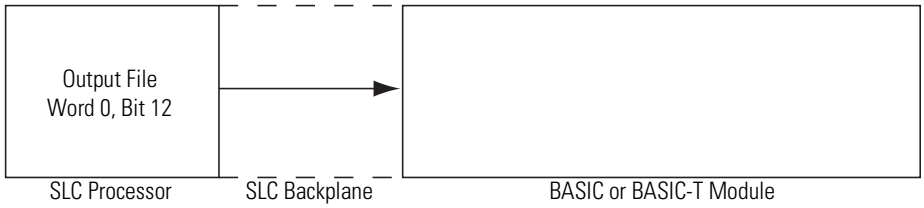
PLC	Notes
-3	For timers and counters, the file number PUSHed (third parameter) is the structure number, limited to a maximum of 255 words.
-5	For timer data, an element is three 16-bit words, stored in the destination file in the following order: Control, Preset, and Accumulator.

If an internal string is chosen, the first character (transaction number) is incremented upon a successful read transaction to inform the module that new data is in the string. The value of the transaction number wraps around from 255 to 0.

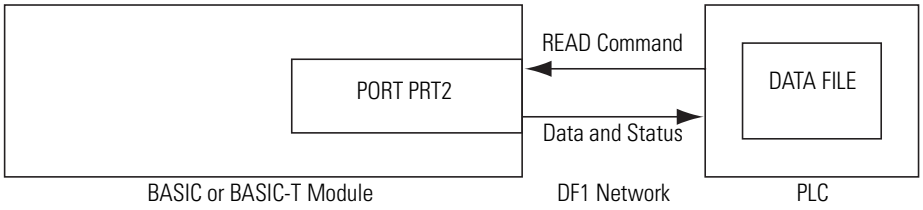
The DF1 port parameters must be set up with CALL 108. The DF1 port can operate with full-duplex or half-duplex slave protocol.

Execute CALL 122 once to set up the data transfer parameters. Input and output image bits (word 0, bit 10) for the slot containing the module, are used to initiate and notify completion of the transfer. The operation is described below:

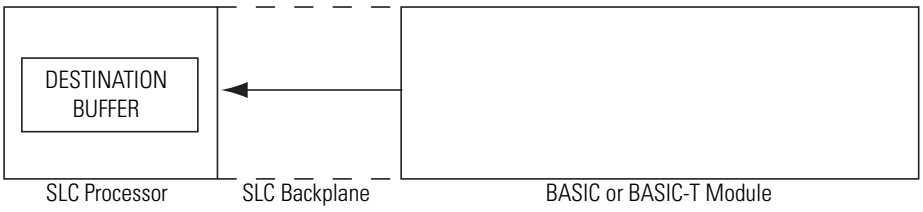
1. The SLC processor sets the output file word 0, bit 10 to inform the module that READ command configured in CALL 122 should be executed.



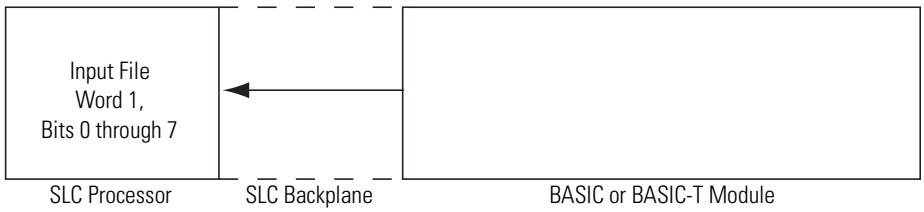
2. The module issues the appropriate READ command to the PLC. The data and status are received from the PLC.



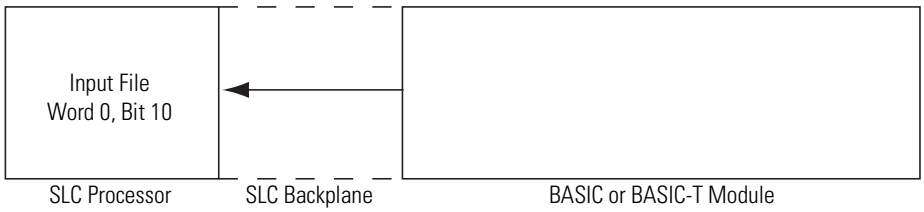
3. When data is available, the module transfers the data into the destination buffer.



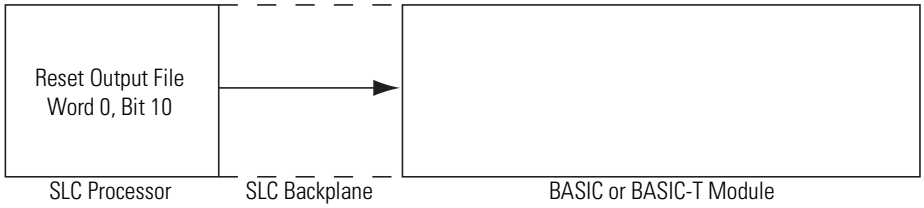
4. The module places the transaction status in the input word 1, bits 0-7.



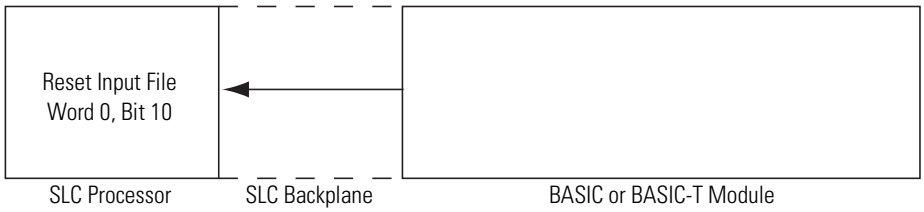
5. The module sets the input file word 0, bit 10 to inform the SLC processor that valid data and status are available.



6. The SLC retrieves the data and status from the buffer and resets output file word 0, bit 10 to inform the module that data was received.



7. The module resets the input file word 0, bit 10 on the same end of scan cycle in which the output file word 0, bit 10 was reset.



The SLC processor must not set, then reset, the output file word 0, bit 10 on the same ladder logic cycle. If this occurs, the module may miss the bit being set.

This CALL is active until it is re-executed with different input parameters.

This CALL has ten input arguments and one output argument.

The first input argument is the type of PLC READ command issued:

- 0 – Disable the previously executed CALL
- 2 – Common interface file - PLC-2 unprotected READ command
- 3 – PLC-3 file - word range READ command
- 5 – PLC-5 file - typed READ command

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.

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 must still be PUSHed.

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 the common interface file is chosen in the first parameter but must still be PUSHed (assumes integer type). If the file type is not one of these, the status equals 2 and the read message does not take place.

Table 13.10 FileType to be Read from Remote Device

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

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.

The sixth input argument is the number of elements to be transferred. If the number is not within the range shown, the status equals 2 and the transfer does not occur.

Table 13.11 Valid Element Length Range

File Type Code	Valid Element Length Range
ASC(N)	1 to 64
ASC(S)	1 to 64
ASC(C)	1 to 21
ASC(T)	1 to 21
ASC(B)	1 to 64
ASC(R)	1 to 21
ASC(I)	1 to 21
ASC(O)	1 to 21
Common Interface File	1 to 21

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, bits 0-7. If the time-out value is not within the range (1 through 255), the status POPped equals 2 and the transfer does not take place.

The eighth input argument is the selection of the destination CPU input image file with or without the internal string, the CPU M1 file with or without the internal string, or the internal string alone:

- 0 – CPU input image file
- 1 – CPU M1 file
- 2 – Internal string
- 3 – CPU input image file and internal string
- 4 – CPU M1 file and internal string

If you chose internal string (2), CALL 29 can be executed to initiate each data transfer without requiring SLC processor interaction. The output file word 0, bit 10 will also initiate a string transaction.

The ninth input argument is the word offset within the destination CPU file. This offset points to the first word transferred. The offset for the internal string is always 1. The first character (transaction number at location 0) is incremented on every successful transfer, to inform the module that new data is in the string. The value of the transaction number wraps around from 255 to 0.

If the CPU input image file is chosen, this offset must not be 0 or 1 because they are reserved words. Zero is reserved for data transfer handshaking bits and word 1 is reserved for the transaction status. A 0 or 1 causes an error to be POPped (2 - bad input parameter) and the CALL is not executed.

If the data received exceeds the string length or CPU file size, the remaining data is truncated.

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 must still be PUSHed.

The output argument is the status of the CALL. It has the following values:

- 0 – Successful
- 1 – Disabled
- 2 – Bad input parameter
- 3 – DF1 not enabled
- 4 – String too small
- 5 – String is not dimensioned

To disable this CALL, a zero must be PUSHed into the first input parameter. All other parameters are ignored but must still be PUSHed.

Whenever an attempt is made to read a remote node, the status of the read is placed into the input word 1, bits 0-7. The possible status codes are shown in Table 13.12.

The status is valid when the module sets the input file word 0, bit 10.

Table 13.12 Transaction Status Codes

Code	Indicates
0	Transfer OK.
1	Transmission failed.
2	Enquiry timeout.
3	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.

Table 13.12 Transaction Status Codes

Code	Indicates
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.
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 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 [selection of destination file]

PUSH [word offset within destination file]

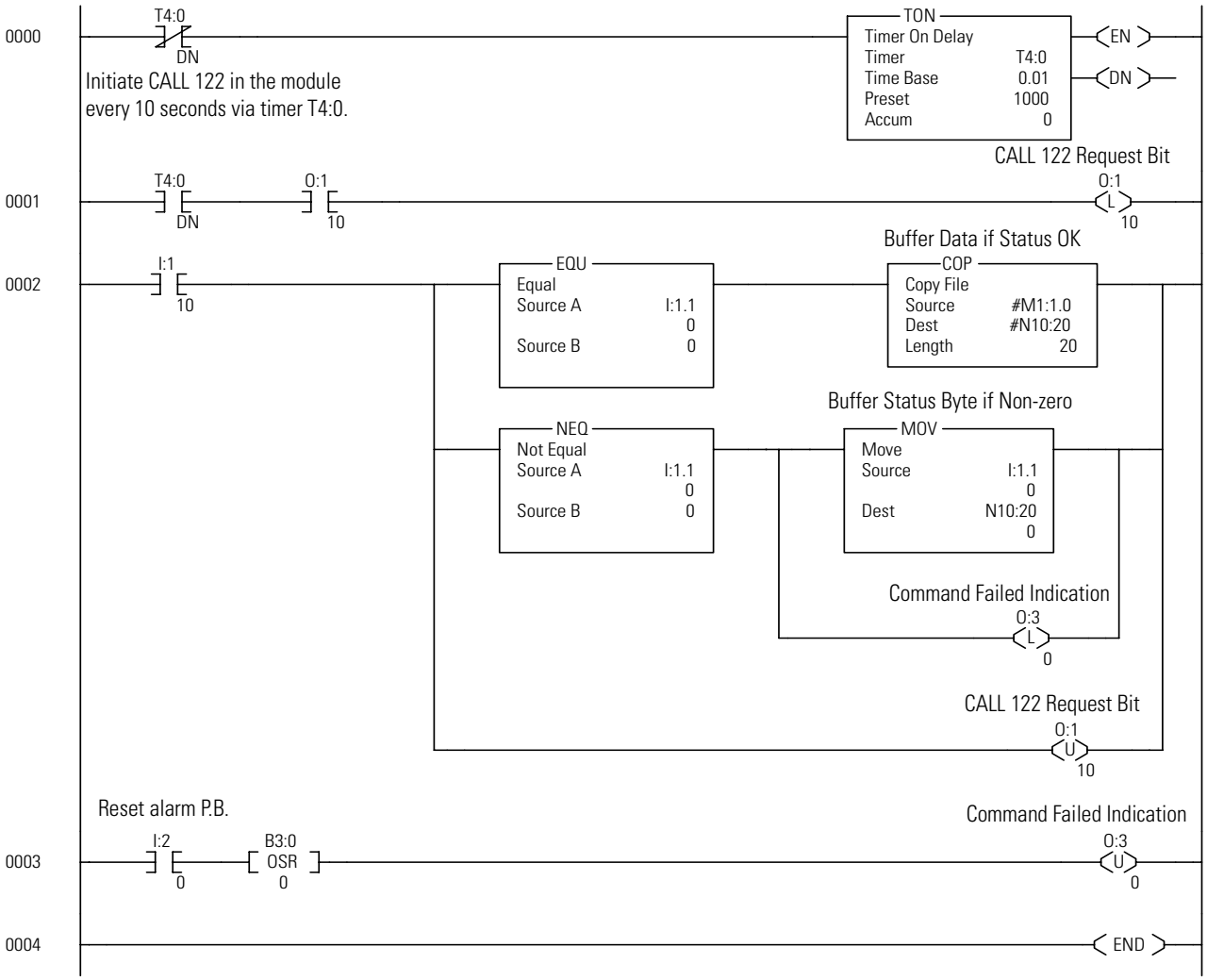
PUSH [string number]

CALL 122

POP [CALL 122 status]

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PLC REMOTE READ COMMAND
>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 SLC M1 FILE
>100 PUSH 0 : REM WORD OFFSET WITHIN M1 FILE
>110 PUSH 0 : REM STRING NUMBER - NOT USED FOR THIS EXAMPLE
>120 CALL 122
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 122 SETUP”
>150 GOTO 150 : REM CALL 122 is active while BASIC program is
running
```



GET

Purpose

Use the GET operator in the Run mode. It 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.

Use the GET# operator to read port PRT2 and the GET@ operator to read port PRT1. The following example prints the decimal representation of any character sent from the console device.

Syntax

GET

Example

```
>1  REM EXAMPLE PROGRAM
>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
>
```

The GET operator 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 the GET operator is placed in the program. There is no buffering of characters on the program port.

INPL

Purpose

Use the INPL statement to read an entire line (up to 254 characters) from 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 the INPL statement.

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

Purpose

Use the INPS statement to read an entire string of characters from the program port buffer. No characters are echoed. The INPS statement is preferred over INPUT 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

Purpose

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 a comma.

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

Examples

```
>INPUT A,C
```

>INPUT A,C causes a question mark (?) to print on the console device. This prompts you to input two numbers separated by a comma. If you do not enter enough data, the module prints **TRY AGAIN** on the console device.

```
>1  REM EXAMPLE PROGRAM
>10 INPUT A,C
>20 PRINT A,C
>RUN

?1

TRY AGAIN

?1,2
l      2

READY
```

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.

```
>1  REM EXAMPLE PROGRAM
>10 INPUT “ENTER A NUMBER”A
>20 PRINT SQR(A)
>30 END

READY
>RUN

ENTER A NUMBER
?4
 2

READY
>

>NEW

>1  REM EXAMPLE PROGRAM
>10 INPUT “ENTER A NUMBER - ”,A
>20 PRINT SQR(A)
>30 END

>RUN
```

```
ENTER A NUMBER - 25
 5

READY
>
```

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.

```
>1 REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 INPUT “NAME(CR),AGE - ”,$(1),A
>30 PRINT “HELLO ”,$(1), “YOU ARE ”,A,“ YEARS OLD.”
>40 END

READY
>RUN

NAME(CR),AGE - PAM
?29
HELLO PAM YOU ARE 29 YEARS OLD.

READY
>
```

You can assign strings and variables with a single INPUT statement.

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,10
>20 INPUT “NAME(CR), AGE - ”,$(1),A
>30 PRINT “HELLO ”,$(1),“, YOU ARE ”, A,“ YEARS OLD”
>40 END
>RUN

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

READY
>
```

LD@

IMPORTANT This instruction is not associated with any port designation.

Purpose

Use the LD@ statement to retrieve floating point numbers that were stored with a ST@ statement. The expression [expr] following the LD@ statement specifies the address where the number is stored after executing the LD@. The LD@ statement places the number on the ARGUMENT STACK at the address location specified by [expr].

This statement can be used with CALL 77 to retrieve variables from a protected area of memory. This protected area is not zeroed on powerup or when the RUN command is issued.

IMPORTANT LD@ is not used with any port designation.

Syntax

LD@ [expr]

Example

```
>P. MTOP
 24515

P. 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 9
>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 9
>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
  24
  25
  26
  27
  28
  29

READY
>PUSH 5FFFH : CALL 77

>P. MTOP
  24575
```

READ

Purpose

Use the READ statement to retrieve the expressions that are specified in the DATA statement 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, they are separated by a comma.

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
>
```

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 considered chained together and appear as one large DATA statement. If at anytime all the data is read and another READ statement is executed, the program terminates and the message **`ERROR: NO DATA - IN LINE XX`** prints to the console device.

Setup Functions

This chapter describes and illustrates commands used to set port parameters within the BASIC program or from the command line. Table 14.1 lists the corresponding mnemonics.

Table 14.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
Set the PRT2 port parameters.	CALL 30	14-1
Set the program port baud rate.	CALL 78	14-2
Reset the print head pointer.	CALL 99	14-3
Reset port PRT1 to the default settings.	CALL 105	14-4
Reset port PRT2 to the default settings.	CALL 119	14-4
Set port parameters of ports PRT1, PRT2, and DH485.	MODE	14-5

CALL 30 – Set PRT2 Port Parameters

Purpose

Use CALL 30 to set the port parameters for port PRT2. Table 14.2 lists the PRT2 port parameters and their selections in the order they are PUSHed on the stack before executing the CALL.

Table 14.2 PRT2 Port Parameters

PRT2 Port Parameters	Selections
Bits per word	5, 6, 7, 8
Parity enable	0 = None, 1 = Odd, 2 = Even
Number of stop bits	1 = 1 Stop bit , 2 = 2 Stop bits , 3 = 1.5 Stop bits
Software handshaking	0 = None, 1 = XON-XOF
Hardware handshaking	0 = Disabled DCD, 1 = Enabled DCD

Syntax

PUSH [bits per word]
PUSH [parity enable]
PUSH [number of stop bits]
PUSH [software handshaking enable/disable]
PUSH [hardware handshaking enable/disable]
CALL 30

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM CALL 30 INPUT PARAMETERS:
>20  REM FIRST PUSH : 5, 6, 7, OR 8 (BITS/CHARACTER)
>30  REM SECOND PUSH : 0, 1, OR 2 (NO PARITY, ODD, OR EVEN)
>40  REM THIRD PUSH : 1, 2, OR 3 (1, 2, OR 1.5 STOP BITS)
>50  REM FOURTH PUSH: 0 OR 1 (SOFTWARE HANDSHAKING
     DISABLE, ENABLED)
>60  REM FIFTH PUSH : 0 OR 1 (HARDWARE HANDSHAKING
     DISABLED, ENABLED)
>70  REM PRT2 DEFAULT CONFIGURATION IS:
>80  REM 1200 BAUD, 8 BITS/CHAR, NO PARITY, 1 STOP BIT, AND
>90  REM SOFTWARE HANDSHAKING ENABLED
>100 PUSH 8 0,1,1,0 : CALL 30
>110 CALL 31

19200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
>
```

CALL 78 – Set Program Port Baud Rate

Purpose

Use CALL 78 to change the program port baud rate from its default value (1200 baud) to one of the following: 300, 600, 1200, 2400, 4800, 9600 or 19200 baud. The default baud rate for the program port is 1200 baud if port PRT1 is configured as the program port or 19200 baud if port DH485 is configured as the program port. PUSH the desired baud rate and CALL 78. The program port remains at this baud rate unless CALL 73 is invoked or the following conditions are met:

- The battery is dead or has been removed.
- The battery-backup capacitor is discharged.
- The EEPROM is removed or not programmed.
- The power is cycled.

If this happens the baud rate defaults to 1200 baud.

Syntax

PUSH [baud rate]
CALL 78

Example

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 4800
>20 CALL 78
```

CALL 99 – Reset Print Head Pointer

Purpose

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.

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 105 – Reset PRT1 to Default Settings

Purpose

Use CALL 105 to reset the port parameters of port PRT1 to their default settings. Table 14.3 lists the default parameters for port PRT1.

Table 14.3 PRT1 Port Parameter Default Settings

PRT1 Port Parameters	Default Setting
Baud rate	1200 baud
Number of data bits	8-bits
Number of stop bits	1-bit
Parity	No parity
Handshaking	Software handshaking

Syntax

CALL 105

Example

```
>1  REM EXAMPLE PROGRAM
>10 CALL 105
```

CALL 119 – Reset PRT2 to Default Settings

Purpose

Use CALL 119 to reset port parameters of PRT2 to their default settings. Table 14.4 lists the default port parameter settings for port PRT2.

Table 14.4 PRT2 Port Parameter Default Settings

PRT2 Port Parameters	Default Setting
Baud rate	1200 baud
Number of data bits	8-bits
Number of stop bits	1-bit
Parity	No parity
Handshaking	Software handshaking

Syntax

CALL 119

Example

```
>1  REM EXAMPLE PROGRAM
>10 CALL 119
```

MODE

Purpose

Use the MODE command to set the port parameters of ports PRT1, PRT2, and DH485.

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 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, 37, 95, and 96 for more information.

The module applies the following rules when hardware handshaking is enabled. The module:

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

Table 14.5 PRT1 and PRT2 Port Parameters

Port Parameters	Selections	Default Settings
Baud rate	300, 600, 1200, 2400, 4800, 9600, 19200	1200
arg1 (parity)	None (N), Even (E), Odd (O)	N
arg2 (number of data bits)	7 or 8	8
arg3 (number of stop bits)	1 or 2	1
arg4 (handshaking)	No handshaking (N) Software handshaking (S) Hardware handshaking (H) Hardware and software handshaking (B)	S
arg5 (storage type)	Store information in user ROM and RAM (E). Store information in battery backed RAM (R).	R

IMPORTANT If any argument (other than port name and baud rate) is left blank, then that argument defaults to the previously specified value for that argument.

Table 14.6 DH485 Port Parameters

Port Parameters	Selections	Default Settings
Baud rate	300, 600, 1200, 2400, 4800, 9600, 19200	19200 baud
arg1 (host node address)	0 to 31	0
arg2 (module node address)	1 to 31	1
arg3 (maximum node address)	1 to 31	31
arg4 (not used)		
arg5 (storage type)	Store information in user ROM and RAM (E). Store information in battery backed RAM (R).	R

IMPORTANT The E storage type option cannot be used if MODE is used as a statement.

Syntax

MODE (port name, baud 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,,,)
```

String Functions

This chapter describes and illustrates commands used to manipulate string data structures within the BASIC program or from the command line. Table 15.1 lists the corresponding mnemonics.

Table 15.1 Chapter Reference Guide

If you need (to)	Use this mnemonic	Page
String repeat	CALL 60	15-1
String append (concatenation)	CALL 61	15-2
Number to string conversion	CALL 62	15-3
String to number conversion	CALL 63	15-4
Find a string in a string.	CALL 64	15-6
Replace a string in a string.	CALL 65	15-7
Insert a string in a string.	CALL 66	15-8
Delete a string from a string.	CALL 67	15-9
Determine the length of a string.	CALL 68	15-10
Allocate memory for strings.	STRING	15-11

CALL 60 – String Repeat

Purpose

Use CALL 60 to repeat a character and place it in a string. You can use the String Repeat when designing output formats. First PUSH the number of times to repeat the character, then PUSH the number of the string containing the repeated character. No arguments are POPped. You cannot repeat more characters than the string's maximum length.

Syntax

PUSH [number of times to repeat character]
PUSH [base string number]
CALL 60

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM STRING REPEAT EXAMPLE PROGRAM
>20 STRING 200,48
>30 $(1) = “*”
>40 PUSH 40 : REM THE NUMBER OF TIMES TO REPEAT CHARACTER
>50 PUSH 1 : REM BASE STRING NUMBER
>60 CALL 60
>70 PRINT $(1)
>80 END

READY
>RUN

**************************************

READY
>
```

CALL 61 – String Append

Purpose

Use CALL 61 to append one string to the end of another string. This CALL expects two string arguments. The first is the string number of the string to be appended and the second is the string number of the base string. If the resulting string is longer than the maximum string length, the append characters are lost. There are no output arguments. This is a string concatenation assignment: (example: **`$(1)=$(1)+$(2)`**).

IMPORTANT If the new string length exceeds the length allocated by the string command, an error message is printed on the console device and the module enters Command mode.

Syntax

PUSH [string number to be appended]
PUSH [base string number]
CALL 61

Example

```
>1   REM EXAMPLE PROGRAM
>10  STRING 200,20
>20  $(1) = “How are ”
>30  $(2) = “you?”
>40  PRINT “BEFORE ”
>50  PRINT “$(1) = ”,$(1)
>60  PRINT “$(2) = ”,$(2)
>70  PUSH 2 : REM STRING NUMBER TO BE APPENDED
>80  PUSH 1 : REM BASE STRING NUMBER
>90  CALL 61 : REM INVOKE STRING APPEND ROUTINE
>100 PRINT “AFTER:”
>110 PRINT “$(1) = ”,$(1)
>120 PRINT “$(2) = ”,$(2)
>130 END

READY
>RUN

BEFORE:
$(1) = How are
$(2) = you?
AFTER:
$(1) = How are you?
$(2) = you?

READY
>
```

CALL 62 – Number to String Conversion

Purpose

Use CALL 62 to convert a number or numeric variable into a string. You must allocate a minimum of 14 characters for the string. If the exponent of the value to be converted is anticipated to be 100 or greater, you must allocate 15 characters. Error checking traps string allocation of less than 14 characters only. There are no output arguments.

Syntax

PUSH [number to convert to string]
PUSH [string number to receive the value]
CALL 62

Example

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,14
>20 INPUT “ENTER A NUMBER TO CONVERT TO A STRING ”,N
>30 PUSH N : REM NUMBER TO CONVERT TO STRING
>40 PUSH 1 : REM CONVERT NUMBER TO STRING 1
>50 CALL 62 : REM DO THE CONVERSION
>60 PRINT $(1)
>70 END

READY
>RUN

ENTER A NUMBER TO CONVERT TO A STRING 2E3
2000

READY
>RUN

ENTER A NUMBER TO CONVERT TO A STRING 1.233
1.233

READY
>
```

CALL 63 – String to Number Conversion

Purpose

Use CALL 63 to convert the first decimal number found in the specified string to a 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.

If the string does not contain a legal value, a zero is returned. A valid value is between 1 and 255. PUSH the number of the string to convert. Two POPs are required. First POP the validity of the value, then POP the actual value. If a string contains a number followed by an E and a letter or non-numeric character, it is assumed that no number was found since the letter is not a valid exponent. (CALL 63 returns a zero in the first argument popped indicating that no valid number was in the string.)

Syntax

PUSH [string number to convert]
CALL 63
POP [validity of the value]
POP [actual value]

Example

```
>1  REM EXAMPLE PROGRAM
>20 INPUT “INPUT A STRING TO CONVERT ”,$(1)
>30 PUSH 1 : REM CONVERT STRING 1
>40 CALL 63
>50 POP V,N
>60 IF (V<>0) THEN PRINT $(1),“ ” N:GOTO 80
>70 PRINT “INVALID OR NO VALUE FOUND”
>80 END

READY
>RUN

INPUT A STRING TO CONVERT 123ABC
123ABC  123

READY
>RUN

INPUT A STRING TO CONVERT 1.2E-7
1.2E-7  1.2 E-7

READY
>RUN

INPUT A STRING TO CONVERT 1.3.6
INVALID OR NO VALUE FOUND

READY
>
```

CALL 64 – Find a String in a String

Purpose

Use CALL 64 to find a string within a string. It locates the first occurrence (position) of this string. This CALL has two input arguments. The first is the string to be found, the second is the string searched for a match. One output argument is required. If the number is not zero then a match was located at the position indicated by the value of the output argument. This routine is similar to the BASIC INSTR$(findstr$,str$) (example: **L=INSTR$($(1),$(2)**).

Syntax

PUSH [string number to be found]
PUSH [base string number]
CALL 64
POP [match position]

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE FIND STRING IN STRING ROUTINE
>20  STRING 100,20
>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 THE 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

READY
>RUN

FOUND AT LOCATION 4

READY
>
```

CALL 65 – Replace a String in a String

Purpose

Use CALL 65 to replace a string within a string. Three arguments are expected. The first argument is the number of the string that replaces the string identified by the second argument string number. The third argument is the base string number. There are no output arguments.

IMPORTANT If the new string length exceeds the length allocated by the string command, an error message is printed on the console device and the module enters Command mode.

Syntax

PUSH [new string number]
PUSH [old string number to be replaced]
PUSH [base string number]
CALL 65

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE OF REPLACE STRING IN STRING
>20  STRING 100,20
>30  $(0) = “RED-LINES”
>40  $(1) = “RED”
>50  $(2) = “BLUE”
>60  PRINT “BEFORE:”
>70  PRINT “$(0) = ”,$(0)
>80  PUSH 2 : REM STRING NUMBER OF THE STRING
     TO BE REPLACED WITH
>90  PUSH 1 : REM STRING NUMBER OF THE STRING TO BE REPLACED
>100 PUSH 0 : REM BASE STRING NUMBER
>110 CALL 65 : REM INVOKE REPLACE STRING IN STRING
>120 PRINT “AFTER:”
>130 PRINT “$(0) = ”,$(0)
>140 END

READY
>RUN

BEFORE:
$(0) = RED-LINES
AFTER:
$(0) = BLUE-LINES

READY
>
```

CALL 66 – Insert a String in a String

Purpose

Use CALL 66 to insert a string within another string. The CALL expects three arguments. The first argument is the position at which to begin the insert. The second argument is the string number of the characters inserted into the base string. The third argument is the number of the base string. This routine has no output arguments.

IMPORTANT If the new string length exceeds the length allocated by the string command, an error message is printed on the console device and the module enters Command mode.

Syntax

PUSH [insert position]
PUSH [string number of inserted character]
PUSH [base string number]
CALL 66

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE ROUTINE TO INSERT A STRING IN A STRING
>20  STRING 100,15
>30  $(0) = “1234590”
>40  $(1) = “67890”
>50  PRINT “BEFORE:”
>60  PRINT “$(0) = ”,$(0)
>70  PUSH 6 : REM POSITION TO START THE INSERT
>80  PUSH 1 : REM STRING NUMBER TO BE INSERTED
>85  PUSH 0 : REM BASE STRING NUMBER
>90  CALL 66 : REM INVOKE INSERT A STRING IN A STRING
>100 PRINT “$(0) = ”, (0)
>110 END

READY
>RUN

BEFORE:
$(0) = 1234590
$(0) = 123456789090

READY
>
```

CALL 67 – Delete a String in a String

Purpose

Use CALL 67 to delete a string from within another string. The CALL expects two arguments. The first argument is the base string number. The second is the number of the string deleted from the base string. This routine has no output arguments.

IMPORTANT This routine deletes only the first occurrence of the string.

Syntax

PUSH [base string number]
PUSH [deleted string number]
CALL 67

Example

```
>1   REM EXAMPLE PROGRAM
>10  REM ROUTINE TO DELETE A STRING IN A STRING
>20  STRING 200,14
>30  $(1) = “123456789012”
>40  $(2) = “12”
>50  PRINT “BEFORE:”
>60  PRINT “$(1) = ”,$(1)
>70  PUSH 1 : REM BASE STRING NUMBER
>80  PUSH 2 : REM STRING NUMBER OF THE STRING DELETED
>90  CALL 67 : REM INVOKE STRING DELETE ROUTINE
>100 PRINT “AFTER:”
>110 PRINT “$(1) = ”,$(1)
>120 END

READY
>RUN

BEFORE:
$(1) = 123456789012
AFTER:
$(1) = 3456789012

READY
>
```

CALL 68 – Find the Length of a String

Purpose

Use CALL 68 to determine the length of a string. One input argument is expected. This is the string number on which the routine acts. One output argument is required. It is the actual number of non-carriage return (CR) characters in this string. This is similar to the BASIC command LEN(str$) (example: **L=LEN($l)**). The length of the string can be properly determined only if the string is terminated with a CR character. If a string is filled using the ASC instruction, a CR must be added as the last character to terminate the string.

Syntax

PUSH [string number]
CALL 68
POP [number of characters]

Example

```
>1  REM EXAMPLE PROGRAM
>10 REM SAMPLE OF STRING LENGTH
>20 STRING 1 0,10
>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

READY
>RUN

THE LENGTH OF 1234567 IS 7

READY
>
```

STRING

Purpose

Use the STRING statement to allocate memory for strings. Initially, no memory is 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 ([expr]) in the STRING statement is the total number of bytes you want to allocate for string storage. The second expression ([expr]) 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.

The module requires one additional byte for each string, plus one additional byte overall. The additional character for each string is allocated for the carriage return character that terminates the string. This means that the statement STRING 100,10 allocates enough memory for 9 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 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 (example: **`NEW`**) and statements (example: **`CLEAR`**) 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.

IMPORTANT The module executes the equivalent of a CLEAR statement every time the STRING [expr],[expr] statement executes. This is necessary because string variables and numeric variables occupy the same external memory space. After the STRING statement executes, all variables are wiped out. Because of this, you should perform string memory allocation early in a program (during the first statement if possible). If you re-allocate string memory you destroy all defined variables.

Syntax

STRING [expr], [expr]

Examples

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,30
>20 $(0) = “-----MONTHLY REPORT-----”
>30 PRINT $(0)

READY
>RUN

-----MONTHLY REPORT-----

READY
>
```

Decimal/Hexadecimal/Octal/ASCII Conversion Table

Mathematical Conversion Overview

The table below lists the decimal, hexadecimal, octal, and ASCII conversions.

Column1				Column 2				Column 3				Column 4			
DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC
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	136	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

BASIC Command, Statement, and CALL Quick Reference Guide

Mnemonic List Overview

The table below lists the various mnemonics found in this manual along with a description and location.

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
ABS( )		Absolute value		3-9
ASC( )		Return integer value of ASCII character.		3-12
ATN( )		Return arctangent of argument.		3-8
BRKPNT		Set program break point.	*	4-2
CALL 14	PUSH [word number of module input buffer] POP [converted value]	16-bit signed integer to BASIC floating-point		9-1
CALL 15	PUSH [word number of module input buffer] POP [converted value]	16-bit unsigned integer to BASIC floating-point		9-2
CALL 16	PUSH [BASIC line number]	Enable the interrupt capability when a DF1 packet is received.		8-2
CALL 17	None	Disable the DF1 packet interrupt capability.		8-3
CALL 18	None	Re-enable the **[CTRL-C]** break function.		4-5
CALL 19	None	Disable the **[CTRL-C]** break function.		4-6
CALL 20	PUSH [BASIC line number]	Enable the SLC processor interrupt capability.		8-3
CALL 21	None	Disable the SLC processor interrupt capability.		8-4
CALL 22	PUSH [source port number] PUSH [maximum number of characters to be transferred] PUSH [decimal value of character delimiter] PUSH [selection of destination file and/or string] PUSH [word offset within the destination file] PUSH [string number] PUSH [byte swap selection] POP [CALL 22 status]	Transfer data from PRT1 or PRT2 to the SLC I/O or M files.		13-2
CALL 23	PUSH [destination port number and/or internal string] PUSH [selection of source file] PUSH [word offset within source file] PUSH [string number] PUSH [byte swap selection] POP [CALL 23 status]	Transfer data from the SLC I/O or M files to PRT1 or PRT2.		12-2

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
CALL 24	PUSH [value to be converted] PUSH [word number of module output buffer]	BASIC floating-point to 16-bit signed integer		9-2
CALL 25	PUSH [value to be converted] PUSH [word number of module output buffer]	BASIC floating-point to 16-bit binary		9-3
CALL 26	POP[SLC processor status]	Generate an interrupt to the SLC processor.		8-4
CALL 27	PUSH [type of 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] PUSH [word offset within destination file] PUSH [string number] POP [CALL 27 status]	Transfer data from a remote DH485 data file to the SLC processor.		13-8
CALL 28	PUSH [type of WRITE command] PUSH [remote node address] PUSH [remote file number] PUSH [remote file type] PUSH [remote starting word offset] PUSH [number of words to be transferred] PUSH [message time-out value] PUSH [selection of source file] PUSH [word offset within source file] PUSH [string number] POP [CALL 28 status]	Transfer data from the SLC processor to a remote DH485 data file.		12-6
CALL 29	PUSH [CALL 27, 28, 122, or 123 for the CALL you want activated] POP [status of transaction]	Handle all errors that are not handled by the ONERR statement.		12-13, 13-13
CALL 30	PUSH [bits per word] PUSH [parity enable] PUSH [number of stop bits] PUSH [software handshaking enable/disable] PUSH [hardware handshaking enable/disable]	Set PRT2 port parameters.		14-1
CALL 31	None	Display current PRT2 port setup.		12-14
CALL 35	POP [ASCII value of character]	Get numeric input character from port PRT2.		13-15
CALL 36	PUSH [buffer selection] POP [number of characters]	Get number of characters in PRT2 buffers.		11-2
CALL 37	PUSH [buffer selection]	Clear port PRT2 input and output buffers.		12-15
CALL 38	PUSH [0 or 1]	Initiate transactions defined by CALLs 27, 28, 122, and 123.		8-5

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
CALL 40	PUSH [hours] PUSH [minutes] PUSH [seconds]	Set clock/calendar time (hour, minute, second).		10-1
CALL 41	PUSH [date] PUSH [month] PUSH [year]	Set clock/calendar date (day, month, year).		10-2
CALL 42	PUSH [day of week]	Set clock/calendar – day of week.		10-3
CALL 43	PUSH [string number]	Retrieve date/time string.		10-4
CALL 44	POP [day] POP [month] POP [year]	Retrieve date numeric (day, month, year).		10-4
CALL 45	PUSH [string number]	Retrieve time string.		10-5
CALL 46	POP [hour] POP [minute] POP [second]	Retrieve time numeric.		10-6
CALL 47	PUSH [string number]	Retrieve day of week string.		10-6
CALL 48	POP [day of week]	Retrieve day of week numeric.		10-7
CALL 51	POP [output image buffer status]	Check CPU output image buffer.		11-3
CALL 52	PUSH [string number]	Retrieve date string.		10-7
CALL 53	POP [processor status]	Transfer CPU output image buffer to module input buffer.		13-17
CALL 54	POP [processor mode]	Transfer module output buffer to CPU input image buffer.		12-15
CALL 55	POP [input image buffer status]	Check CPU input image buffer.		11-4
CALL 56	PUSH [number of words to be transferred] POP [processor status]	Transfer CPU M0 file to module input buffer.		13-18
CALL 57	PUSH [number of words to be transferred] POP [processor mode]	Transfer module output buffer to CPU M1 file.		12-16
CALL 58	POP [module file M0 write status]	Check M0 file status.		11-5
CALL 59	POP [module file M1 read status]	Check M1 file status.		11-6
CALL 60	PUSH [number of times to repeat character] PUSH [base string number]	String repeat		15-1
CALL 61	PUSH [string number to be appended] PUSH [base string number]	String append (concatenation)		15-2
CALL 62	PUSH [number to convert to string] PUSH [string number to receive the value]	Number to string conversion		15-3
CALL 63	PUSH [string number to convert] POP [validity of the value] POP [actual value]	String to number conversion		15-4

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
CALL 64	PUSH [string number to be found] PUSH [base string number] POP [match position]	Find a string in a string.		15-6
CALL 65	PUSH [new string number] PUSH [old string number to be replaced] PUSH [base string number]	Replace a string in a string.		15-7
CALL 66	PUSH [insert position] PUSH [string number of inserted character] PUSH [base string number]	Insert string in a string.		15-8
CALL 67	PUSH [base string number] PUSH [deleted string number]	Delete string from a string.		15-9
CALL 68	PUSH [string number] POP [number of characters]	Determine length of a string.		15-10
CALL 70	None	ROM to RAM program transfer		8-8
CALL 71	PUSH [ROM program number]	ROM/RAM to ROM program transfer		8-9
CALL 72	None	RAM/ROM return		8-9
CALL 73	None	Battery-backed RAM disable		5-1
CALL 74	None	Battery-backed RAM enable		5-2
CALL 75	POP [processor mode]	Check SLC 500 controller CPU status.		11-7
CALL 77	PUSH [new MTOP address]	Protected variable storage		5-2
CALL 78	PUSH [baud rate]	Set program port baud rate.		14-2
CALL 80	POP [battery status]	Check battery condition.		11-8
CALL 81	None	User memory module check and description	*	5-3
CALL 82	None	Check user memory module map.	*	5-4
CALL 84	PUSH [starting word offset in DH485 interface file] PUSH [number of words to be transferred] POP [transfer status]	Transfer DH485 interface file to module input buffer.		13-19
CALL 85	PUSH [starting word offset in DH485 interface file] PUSH [number of words to be transferred] POP [transfer status]	Transfer module output buffer to DH485 interface file.		12-17
CALL 86	POP [DH485 interface file remote write status]	Check DH485 interface file remote Write status.		11-8
CALL 87	POP [DH485 interface file remote read status]	Check DH485 interface file remote Read status.		11-9
CALL 88	PUSH [number to convert] PUSH [output buffer to receive converted value]	Convert BASIC floating-point to SLC floating-point		9-4
CALL 89	PUSH [input buffer of value to be converted]	Convert SLC floating-point to BASIC floating-point		9-5

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
CALL 90	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] POP [status of message instruction]	Read remote DH485 data file to BASIC input buffer.		13-20
CALL 91	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] POP [status of message instruction]	Write module output buffer to remote DH485 data file.		12-18
CALL 92	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] POP [status of message instruction]	Read remote DH485 interface file to module input buffer.		13-23
CALL 93	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] POP [status of message instruction]	Write module output buffer to remote DH485 interface file.		12-22
CALL 94	None	Print current PRT1 port setup.		12-24
CALL 95	PUSH [buffer selection] POP [number of characters]	Get number of characters in PRT1 buffers.		11-10
CALL 96	PUSH [buffer selection]	Clear port PRT1 input and output buffers.		12-24
CALL 97	None	Enable port PRT2 DTR signal.		11-11
CALL 98	None	Disable port PRT2 DTR signal.		11-11
CALL 99	None	Reset print head pointer.		14-3
CALL 101	PUSH [starting address] PUSH [ending address]	Upload user memory module code to host.		5-4
CALL 103	None	Print port PRT1 output buffer and pointer.		5-5
CALL 104	None	Print port PRT1 input buffer and pointer.		5-6
CALL 105	None	Reset port PRT1 to default settings.		14-4

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
CALL 108	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 [module DF1 address]	Enable DF1 driver communications.		11-12
CALL 109	None	Print the argument stack.		5-7
CALL 110	None	Print port PRT2 output buffer and pointer.		5-8
CALL 111	None	Print port PRT2 input buffer and pointer.		5-8
CALL 112	PUSH [LED1 state] PUSH [LED2 state]	User LED control		12-25
CALL 113	None	Disable DF1 driver communications.		11-18
CALL 114	None	Transmit DF1 packet.		12-26
CALL 115	POP [DF1 transmit status]	Check DF1 XMIT status.		12-27
CALL 117	POP [length of DF1 packet]	Get DF1 packet length.		13-25
CALL 118	PUSH [CALL enable/disable] PUSH [selection of destination file and/or string] PUSH [word offset in destination file] PUSH [string number] PUSH [maximum word length] POP [CALL 118 status]	Allow unsolicited writes from a remote SLC or PLC node.		13-26
CALL 119	None	Reset port PRT2 to default settings.		14-4
CALL 120	PUSH [decimal equivalent]	Clear module input and output buffers.		11-18
CALL 121	POP [program ID number]	Get SLC processor program ID number.		11-19
CALL 122	PUSH [type of 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 [selection of destination file] PUSH [word offset within destination file] PUSH [string number] POP [CALL 122 status]	Read a PLC data file.		13-30

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
CALL 123	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 source file] PUSH [word offset within source file] PUSH [string number] POP [CALL 123 status]	Write to a PLC data file.		12-28
CBY( )		Retrieve data from specified memory address.		3-16
CHR( )		Count value converted ASCII character.		3-14
CLEAR		Clear variables, interrupts & strings.		6-1
CLEARI		Clear interrupts.		6-3
CLEARS		Clear all stacks.		6-3
CLOCK0		Disable real time clock.		7-2
CLOCK1		Enable real time clock.		7-1
CONT		Continue after a Stop or **[CTRL-C]**.	*	4-3
CONTROL-C		Stop execution & return to Command mode.		4-4
CONTROL-Q		Restart a list after **[CTRL-S]**.		4-8
CONTROL-S		Interrupt a list command.		4-7
COS( )		Return the cosine of argument.		3-8
DATA		Data read by Read statement.		6-4
DBY( )		Retrieve or assign data to or from the internal data memory of the module.		3-16
DIM		Allocate memory for array variables.		6-4
DO-UNTIL		Set up conditional do-loop.		7-4
DO-WHILE		Set up a conditional do-loop.		7-3
EDIT		Edit a line of the BASIC program.	*	4-8
END		Terminate program execution.		7-5
EOF		Test for empty input buffer.		3-15
ERASE		Delete the last BASIC program stored in ROM by a PROG command.	*	4-9
EXP( )		“e” (2.7182818) TO THE X		3-11
FOR-TO-(STEP)-NEXT		Set up for-next loop.		7-6
FREE		Test for number of free bytes of RAM memory.		3-15

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
GET		Read console input device.		13-38
GET#		Read console input device connected to PRT2.		13-38
GET@		Read console input device connected to PRT1.		13-38
GOSUB		Execute subroutine.		8-11
GOTO		Go to program line number.		7-7
IDLE		Force module to enter "wait until interrupt mode".		4-10
IF-THEN-ELSE		Conditional test		7-8
INPL		Read line of characters from the program port buffer.		13-39
INPL#		Read line of characters from port PRT2 buffer.		13-39
INPL@		Read line of characters from port PRT1 buffer.		13-39
INPS		Read string of characters from the program port buffer.		13-40
INPS#		Read string of characters from port PRT2 buffer.		13-39
INPS@		Read string of characters from port PRT1 buffer.		13-39
INPUT		Input a string or variable.		13-40
INPUT#		Input a string or variable from port PRT2.		13-40
INPUT@		Input a string or variable from port PRT1.		13-40
INT( )		Integer		3-10
LD@		Load variable.		13-43
LEN		Read the number of bytes of memory in the current selected program.		3-15
LET		Assign a variable or string a value (LET is optional).		6-5
LIST		List program to the console devise.	*	4-11
LIST#		List program to serial printer.	*	4-12
LIST@		List program to device connected to port PRT1.	*	4-12
LOG( )		Natural log		3-11
MODE		Set port parameters of ports PRT1, PRT2, and DH485.		4-12, 14-5
MTOP		Read the last valid memory address.		3-16
NEW		Erase the program stored in RAM.	*	4-14
NEXT		Test for-next loop condition.		7-9
NOT( )		One's complement		3-9
NULL		Set NULL count after carriage return-line feed.	*	4-14

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
ONERR		Go to line number when an error is detected.		8-12
ON-GOSUB		Conditional GOSUB		8-14
ON-GOTO		Conditional GOTO		7-11
ONTIME		Generate an interrupt when TIME is equal to or greater than ONTIME argument-line number.		8-14
PH0.		Print hex value with zero suppression to console device.		12-37
PH0.#		Print hex value with zero suppression to PRT2.		12-37
PH0.@		Print hex value with zero suppression to PRT1.		12-37
PH1.		Print hex value with no zero suppression to console device.		12-37
PH1.#		Print hex value with no zero suppression to PRT2.		12-37
PH1.@		Print hex value with no zero suppression to PRT1.		12-37
PI		PI-3.1415926		3-10
POP		POP argument stack to variables.		8-17
PRINT		Print variables, strings or literals to console device; P. is shorthand for print.		12-35
PRINT#		Print to port PRT2.		12-35
PRINT@		Print to port PRT1.		12-35
PRINT CR		Print carriage return.		12-36
PRINT SPC( )		Print spaces.		12-36
PRINT TAB( )		Print tabs.		12-36
PRINT USING(Fx)		Print numeric values in scientific notation.		12-36
PRINT USING(#.#)		Print numeric values in decimal notation.		12-36
PRINT USING(0)		Restore the default print mode.		12-37
PROG		Save the current program in EPROM.	*	4-15
PROG1		Save baud rate information in EPROM.	*	4-16
PROG2		Save baud rate information in EPROM and execute program after reset.	*	4-17
PUSH		PUSH expressions on argument stack.		8-15
RAM		Evoke RAM mode.	*	4-19
READ		READ data in data statement.		13-45

Mnemonic	Required PUSHes or POPs	Description	Active In Command Mode Only	Page
REM		Specify a remark or comment line.		4-19
REN		Renumber BASIC program.		4-20
RESTORE		RESTORE read point.		6-7
RETI		Return from interrupt.		8-18
RETURN		RETURN from subroutine.		8-18
RND		Random number		3-11
ROM		Select ROM mode.	*	4-20
RROM		Select ROM mode and execute the selected program.	*	4-21
RUN		Execute a program.	*	4-22
SGN		Sign		3-10
SIN( )		Return the sine of argument		3-8
SNGLSTP		Initiate single-step program execution.	*	4-23
SQR( )		Square Root		3-10
ST@		Store variable.		12-38
STOP		Break program execution.		8-20
STRING		Allocate memory for strings.		15-11
TAN( )		Return the tangent of the argument.		3-8
TIME		Retrieve and/or assign free running clock value.		3-17
VER		Verify module firmware version.		4-25
XBY( )		Retrieve or assign data to or from the external data memory of the module.		3-17
XFER		Transfer a program from ROM to RAM.	*	4-26
+		Addition		3-3
/		Division		3-4
**		Exponentiation		3-4
*		Multiplication		3-4
-		Subtraction		3-4
.AND.		Logical AND		3-6
.OR.		Logical OR		3-6
.XOR.		Logical Exclusive OR		3-6
@		Direct communications to port PRT1.		3-15
#		Direct communications to port PRT2.		3-15

Symbols

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Reach us now at www.rockwellautomation.com

Wherever you need us, Rockwell Automation brings together leading brands in industrial automation including Allen-Bradley controls, Reliance Electric power transmission products, Dodge mechanical power transmission components, and Rockwell Software. Rockwell Automation's unique, flexible approach to helping customers achieve a competitive advantage is supported by thousands of authorized partners, distributors and system integrators around the world.

Americas Headquarters, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
European Headquarters SA/NV, avenue Herrmann Debroux, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1746-RM001A-US-P - April 2000
Supercedes Publication 1746-6.3 - November 1994