

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Rockwell / Allen-Bradley 1771-DB* at our website: ***Click HERE***

5.5.2 Unary Operators (continued)

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 (i.e. greater than 1000). This is because significant digits, in the decimal (fraction) portion of 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.

5.5.3 Understanding Precedence of Operators

You can write complex expressions using only a small number of parenthesis. To illustrate the precedence 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 because multiplication has higher precedence. Use parentheses if you are in doubt about the order of precedence. The precedence of operators from highest to lowest in the BASIC Module is:

1) Operators that use parenthesis ()
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.)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.5.4 How Relational Expressions Work

Relational expressions involve the operators =, < >, > , > =, < , and < =. These operators are typically used to "test" a condition. In the BASIC Module relational operations are typically used to "test" a condition. In the BASIC Module relational operators return a result of 665535 (OFFFFH) if the relational expression is true, and a result of 0, if the relation expression is false. The result is returned to the argument stack. Because of this, it is possible to display the result of a relational expression.

Examples:

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

0 65535 0 65535
```

You can "chain" relational expressions together using the logical operators .AND., .OR., and .XOR.. This makes it possible to test a complex condition with ONE statement.

Example:

```
> 10 IF ([A > B].AND. [A > C]) .OR.[A > D] THEN
```

Additionally, you can use the NOT([expr]) operator.

Example:

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

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.6 Special Operators

Special operators include special function operators and system control values.

5.6.1 Special Function Operators

Special function operators directly manipulate the I/O hardware and the memory addresses on the processor.

5.6.1.1 GET

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. The following example prints the decimal representation of any character sent from the console:

Example:

```
> 10 A=GET
> 20 IF A < > 0 THEN PRINT A :REM ZERO MEANS NO ENTRY
> 30 GOTO 10
> RUN
  65    (TYPE "A" ON CONSOLE)
  49    (TYPE "1" ON CONSOLE)
  24    (TYPE "CONTROL–X" ON CONSOLE)
  50    (TYPE "2" ON CONSOLE)
```

The GET operator 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.

5.6.1.2 TIME

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.6.1 Special Function Operators (continued)

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

Example:

```
> CLOCK1      (enable REAL TIME CLOCK)
> CLOCK0      (disable REAL TIME 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. Note that each count in internal memory location 71 (47H) represents 5 milliseconds of TIME.

Continuing with the example:

```
> DBY(71) = 0                (fraction of TIME = 0)
> PRINT TIME
  0
> DBY(71) = 3                (fraction of TIME = 3, 15 ms)
> PRINT TIME
  1.5 E–2
```

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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.6.2 System Control Values

The system control values determine how memory is allocated by the BASIC Module.

5.6.2.1 MTOP

After reset, the BASIC Module sizes the external memory and assigns the last valid memory address to the system control value, MTOP. The module does not use any external RAM memory beyond the value assigned to MTOP.

Examples:

```
> PRINT MTOP          or      PH0. MTOP
14335                           37FFH
```

5.6.2.2 LEN

The system control value, LEN, tells 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, variables and array memory usage. You cannot assign LEN a value, it can only be read. A NULL program (i.e. no program) returns a LEN of 1. The 1 represents the end of program file character.

Important: The BASIC Module does not require any "dummy" arguments for the system control values.

5.7 Data Transfer Support Routines

The BASIC Module communicates with the host processor using block-transfer communications. The host processor sends variable length blocks of data to the module. You can transfer a maximum of 64 words in and 64 words out per scan. The module responds with the requested block length of data. The module has an auxiliary processor dedicated to servicing of the block-transfers to and from the host processor. These support routines provide the communications link between the host processor and the BASIC processor. The block-transfer-read (BTR) buffer resides at address 7800H and is 128 bytes long. You can examine the values in the buffer using the XBY command, i.e. PRINT XBY(7800H).

The block-transfer-write (BTW) buffer resides at one of two starting addresses; 7B00H or 7C00H . Examine addresses 7D0 AH and 7D0BH which point to the proper starting address. 7D0AH contains the high byte of the address and 7D0BH contains the low byte of the address. The BTW buffer is also 128 bytes long. You can also examine these values in the buffer using the XBY command.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.7 Data Transfer Support Routines (continued)

Example:

```
> 10 REM PRINT CONTENTS OF BTW BUFFER
> 20 C=(XBY(7D0AH)*100H)+XBY(7D0BH)
> 30 FOR I=0 TO 128
> 40 PH0. XBY(C+I),
> 50 NEXT I
> 60 END
```

5.7.1 Update Block-Transfer-Read Buffer (timed) – CALL 2

This routine transfers the block-transfer-read (BTR) buffer to the auxiliary processor on the BASIC Module for use in the next BTR request from the host processor. If no data transfer occurs within 2 seconds the routine returns to BASIC without transferring data. This routine has no input arguments and one output argument (the status of the transfer). A non-zero returned means that no transfer occurred and the CALL timed out. A zero returned means a successful transfer.

Example:

```
> 10 CALL 2
> 20 POP X
> 30 IF X < > 0 PRINT "TRANSFER UNSUCCESSFUL"
```

5.7.2 Update Block-Transfer-Write Buffer (timed) – CALL 3

This routine transfers the block-transfer-write (BTW) buffer of the auxiliary processor on the BASIC Module to the BASIC BTW buffer. If no data transfer occurs within 2 seconds the routine returns to BASIC with no new data. This routine has no input arguments and one output argument (the status of the transfer). A non-zero returned means that no transfer occurred and the CALL timed out. A zero returned means a successful transfer.

Example:

```
> 10 CALL 3
> 20 POP X
> 30 IF X < > 0 PRINT "TRANSFER UNSUCCESSFUL"
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.7.3 Set Block-Transfer-Write Length – CALL 4

This routine sets the number of words (1–64) to transfer between the BASIC Module and the host processor. The processor program block-transfer length must match the set value. Only one argument is input (the number of words to BTW) and none returned. If not used, the default length is 5 words.

Example:

```
> 10 PUSH 10:
> 20 CALL 4
```

5.7.4 Set Block-Transfer-Read Length – CALL 5

This routine sets the number of words (1–64) to transfer between the BASIC Module and the host processor. The processor program block-transfer length must match the set value. Only one argument is input (the number of words to BTR) and none returned. If not used, the default length is 5 words.

Example:

```
> 10 PUSH 10
> 20 CALL 5
```

5.7.5 Update Block-Transfer-Write Buffer – CALL 6

This routine transfers the block-transfer-write (BTW) buffer of the auxiliary processor on the BASIC Module to the BASIC BTW buffer. This routine waits until a block-transfer occurs if one has not previously occurred.

5.7.6 Update Block-Transfer-Read Buffer – CALL 7

This routine transfers the block-transfer-read (BTR) buffer to the auxiliary processor on the BASIC Module for use in the next BTR request from the host processor. This routine waits until a block-transfer occurs.

5.7.7 Disable Interrupts – CALL 8

Important: You must use an Interrupt Control CALL 8 before a PROG command to disable interrupts.

This routine disables system interrupts. It is mandatory for PROM programming. The wall clock cannot be accessed and the peripheral port is disabled by this call. Some call routines enable interrupts automatically upon completion.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.7.8 Enable Interrupts – CALL 9

Important: You must use an Interrupt Control CALL 9 after a PROG command to re-enable interrupts.

This routine enables system interrupts. It is mandatory after programming a PROM. The wall clock is accessible and the peripheral port is enabled.

5.7.9 Input Call Conversion Routines

All of the input call conversion routines require the same sequence of commands. These are:

PUSH – the word position of the processor block-transfer-file to be converted (1–64)

CALL – the appropriate input conversion

POP – the input argument into a variable

The data format of the output arguments from each of the input conversion routines is described in more detail in Chapter 7.

5.7.9.1 3-Digit Signed, Fixed Decimal BCD to Internal Floating Point ± XXX – CALL 10

The input argument is the number (1 to 64) of the word in the write-block-transfer buffer that is to be converted from 3-digit BCD to internal format. The output argument is the converted value in internal floating point format. The sign bit is bit number 16.

5.7.9.2 16-Bit Binary (4-digit hex) to Internal Floating Point – CALL 11

The input argument is the number (1 to 64) of the word in the write-data-transfer buffer to be converted from 16-bit binary to internal format. The output argument is the converted value in internal floating point format. There are no sign or error bits decoded.

5.7.9.3 4-Digit Signed Octal to Internal Floating Point ± XXXX – CALL 12

The input argument is the number (1 to 64) of the word in the processor block-transfer buffer to be converted from 4-digit signed octal to internal format. This 12-bit format has a maximum value of ± 7777 octal. The output argument is the converted value in internal floating point format. The sign bit is bit number 16.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.7.9 Input Call Conversion Routines (continued)

5.7.9.4 6-Digit, Signed, Fixed Decimal BCD to Internal Floating Point ± XXXXXX – CALL 13

The input argument is the number (1–64) of the first word (6-digit BCD is sent to the BASIC module in two processor words) of the write-block-transfer buffer to be converted from 6-digit, signed, fixed decimal BCD to internal format. The maximum values allowed are ± 999999. The output argument is the converted value in internal floating point format. The sign bit is bit number 16.

5.7.9.5 4-Digit BCD to Internal Floating Point XXXX – CALL 17

The input argument is the number (1–64) of the word in the write-block-transfer buffer to be converted from 4-digit BCD to internal format. The maximum value allowed is 0–9999. The output argument is the converted value in internal floating point format.

Sample input conversions:

```
> 20 PUSH 3 :REM CONVERT 3rd WORD OF PLC DATA
> 30 CALL 10 :REM DO 3 DIGIT BCD TO F.P. CONVERSION
> 40 POP W :REM GET CONVERTED VALUE – STORE IN VARIABLE W
> 20 PUSH 9 :REM CONVERT STARTING WITH 9th WORD OF PLC DATA
> 30 CALL 13 :REM DO 6 DIGIT BCD TO F.P. CONVERSION
> 40 POP L(9) :REM GET CONVERTED VALUE – STORE IN ARRAY L (9)
```

5.7.10 Output Call Conversion Routines

All of the output call conversion routines require the same sequence of commands. These are:

PUSH – the value to be converted

PUSH – the word position of the processor block-transfer-file to be converted (1–64)

CALL – the appropriate output conversion

The data format of the converted value of each of the output conversion routines is described in more detail in Chapter 7. Use these output call conversion routines to load the BTR buffer before executing a CALL 7 or CALL 2.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.7.10 Output Call Conversion Routines (continued)

5.7.10.1 Internal Floating Point to 3-Digit, Signed, Fixed Decimal BCD ± XXX – CALL 20

This routine has two input arguments and no output arguments. The first argument is the variable with a value in the range of –999 to +999 that is converted to a signed 3-digit binary coded decimal format used by the processor. The second input argument is the number of the word to receive the value in the read-block-transfer buffer. The sign bit is bit number 16.

Example:

```
> 20 PUSH W  :REM DATA TO BE CONVERTED
> 30 PUSH 6  :REM WORD LOCATION TO GET DATA
> 40 CALL 20
```

5.7.10.2 Internal Floating Point to 16-Bit Unsigned Binary (4 digit hex) – CALL 21

This routine takes a value between 0 and 65535 and converts it to its binary representative and stores this in the read-block-transfer buffer in one word. Two arguments are PUSHed and none POPed. The first value PUSHed is the data or variable. This is followed by the number of the word to receive the value in the read-block-transfer buffer.

Example:

```
> 50 PUSH T  :REM THE VALUE TO CONVERTED TO 16 BINARY
> 60 PUSH 3  :REM WORD 3 IN THE BTR BUFFER GETS THE VALUE T
> 70 CALL 21 :REM DO THE CONVERSION
```

5.7.10.3 Internal Floating Point to 4-Digit, Signed Octal ± XXXX–Call 22

This routine converts a value from internal format to a four digit signed octal value. Two arguments ar PUSHed and non POPed. The first value PUSHed is the data ($\pm 7777_8$) or variable. This is followed by the number of the word to receive the value in the read-block-transfer buffer. The sign bit is bit number 16.

Example:

```
> 50 PUSH H  :REM THE VALUE TO CONVERTED TO 4-DIGIT SIGNED OCTAL
> 60 PUSH 3  :REM WORD 3 IN THE BTR BUFFER GETS THE VALUE H
> 70 CALL 22 :REM DO THE CONVERSION
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.7.10 Output Call Conversion Routines (continued)

5.7.10.4 Internal Floating Point to 6-Digit, Signed, Fixed Decimal BCD ± XXXXXX – CALL 23

This routine converts an internal 6-digit, signed, integer to a 2 word format and places the converted value in the read-block-transfer buffer. Two arguments are PUSHed and none POPed. The first value PUSHed is the data or variable. This is followed by the number of the word to receive the value in the read-block-transfer buffer. The sign bit is bit number 16.

Example:

```
> 20 PUSH 654321. OR
> 30 PUSH 3
> 40 CALL 23
```

```
> 20 PUSH B(I)
> 30 PUSH 3
> 40 CALL 23
```

5.7.10.5 Internal Floating Point to 3.3-digit, Signed, Fixed Decimal BCD ± XXX.XXX – CALL 26

This routine converts a variable in internal format to a signed, 6-digit, fixed decimal point number which is stored in 2 words in the block-transfer-read buffer. Two arguments are PUSHed and none POPed. The first value PUSHed is the data (± 999.999) or variable. This is followed by the number of the first word to receive the value in the block-transfer-read buffer. The sign bit number is 16.

Example:

```
> 50 PUSH S      :REM THE VALUE TO BE CONVERTED TO 3.3-DIGIT FIXED POINT
> 60 PUSH 3      :REM WORD 3 IN THE BTR BUFFER GETS THE VALUES
> 70 CALL 26     :REM DO THE CONVERSION
```

5.7.10.6 Internal Floating Point to 4-digit BCD XXXX – CALL 27

This routine converts a value in internal floating point format to a 4-digit, unsigned BCD value and places it in the read-block-transfer buffer. Two arguments are PUSHed and none POPed. The first value PUSHed is the data (0–9999) or variable. This is followed by the number of the word to receive the value in the read-block-transfer buffer.

Example:

```
> 20 PUSH B      REM THE VALUE TO BE CONVERTED TO 4-DIGIT BCD
> 30 PUSH 7      :REM WORD 7 IN THE BTR BUFFER GETS THE VALUE B
> 40 CALL 27     :REM DO THE CONVERSION
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8 Peripheral Port Support

The peripheral port is used for:

1. exchanging data with an external device with user written protocol.
2. listing programs with the LIST# or LIST @ statement.
3. loading/saving programs using the 1771–SA/SB data recorders.

For these functions to operate properly and to allow flexibility for talking to many different devices we provide parameter setup routines.

To make best use of the port, a 256 character buffer is provided on both input and output. This means that the module may receive up to 256 characters without error from a device before the module must service the data. The module can store an entire response to a command in the input buffer. This is useful when the data arrives in a high speed burst that is too fast for the BASIC program to handle character by character.

The 256 character output buffer allows the module to assemble an entire message leaving your user program free to perform other operations while the message is sent character by character by the device driver.

5.8.1 Peripheral Port Support – Parameter Set – CALL 30

This routine sets up the parameters for the peripheral port. The parameters set are the number of bits/word, parity enable or disable, even or odd parity, number of stop bits, software handshaking and hardware handshaking. Default values are:

- 1 stop bit
- no parity
- 8 bits/character
- DCD off
- XON/XOFF disabled
- 1200 baud (jumper selected)
- 1 start bit (fixed)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.1 Peripheral Port Support – Parameter Set – CALL 30 (continued)

The five values PUSHed on the stack before executing the CALL are in the following order:

Parameter	Selections
Number of bits/word	5,6,7,8
Parity Enable	0=None,2=Even 1=Odd
Number of Stop Bits	1=1 Stop Bit, 2=2 Stop Bits, 3=1.5 Stop Bits
Software Handshaking	0=None, 1=XON–XOF
Hardware Handshaking	0=Disable DCD
	1=enable DCD

```
Example:  100 PUSH 8 : REM 8 BITS/WORD
          120 PUSH 0 : REM NO PARITY
          140 PUSH 1 : REM 1 STOP BIT
          160 PUSH 0 : REM NO SOFTWARE HANDSHAKING
          180 PUSH 0 : REM DISABLE DCD
          200 CALL 30 : REM SET UP PERIPHERAL PORT
or
Example:  100 PUSH 8,0,1,0,0:CALL 30
```

Important: Hardware and software handshaking is disabled and both I/O buffers are cleared on power-up.

5.8.2 Peripheral Port Support – Display Peripheral Port Parameters –CALL 31

This routine displays the current peripheral port configuration on the terminal. No argument is PUSHed or POPed.

Enter CALL 31 [return] from the command mode.

Example:

```
CALL 31 [RETURN]

DCD OFF
1 STOP BIT
NO PARITY
8 BITS/CHAR
```

Important: XON/XOFF status is shown only if enabled.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.3 Save Program to Data Recorder – CALL 32

Important: Maximum baud rate is 2400 bps for all data recorder CALL routines. The Peripheral Port is set up in these routines and does not need setting in BASIC.

This routine saves the current RAM program on a 1770–SA/SB recorder. The program in RAM is not modified. The message ERROR—IN ROM displays if ROM is selected instead of RAM.

To use this routine:

- insert the cassette/cartridge into the recorder and rewind.
- enter CALL 32. The terminal responds with "POSITION TAPE AND PRESS WRITE/RECORD ON TAPE".
- Press RECORD. The module begins sending data to the recorder. An asterisk is displayed on the terminal as each program line is sent to the recorder.

During this time the "DATA IN" LED on the data recorder and the XMIT LED on the front of the module illuminate. When the last line of program is sent to the recorder, a final asterisk is displayed on the terminal indicating that the save operation is complete. Finally, tape motion ceases and the BASIC prompt (>) is displayed.

- Press STOP if you are using a 1770–SA Recorder.

Important: If there is no program in RAM, the module displays the message ERROR–NO BASIC PROGRAM EXISTS.

5.8.4 Verify Program with Data Recorder – CALL 33

Important: Maximum baud rate is 2400 bps for all data recorder CALL routines. The Peripheral Port is set up in these routines and does not need setting in BASIC. See section 4.2.2.2 for required cable connections.

This routine is used to verify the current RAM program with a previously stored program on the data recorder. To use this routine:

- insert the cassette cartridge into the recorder and rewind.
- enter CALL 33. The terminal responds with "POSITION TAPE AND PRESS READ FROM TAPE".

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.4 Verify Program with Data Recorder – CALL 33 (continued)

- Press PLAY. Tape movement begins and the recorder searches for the beginning of the next program. As each line of the program verifies, an asterisk displays.

  During this time the "DATA OUT" LED on the recorder and the "RECV" LED on the module illuminate. When the last line of program is verified a final asterisk is displayed on the terminal followed by the "VERIFICATION COMPLETE" message and the BASIC (>) prompt. If any differences between programs are encountered, the routine displays an "ERROR–VERIFICATION FAILURE" message and the BASIC (>) prompt is immediately displayed.

5.8.5 Load Program from Data Recorder – CALL 34

Important: Maximum baud rate is 2400 bps for all data recorder CALL routines. The UART is set up in these routines and does not need setting in BASIC. See section 4.2.2.2 for required cable connections.

This routine loads a program stored on a 1770–SA/SB recorder into user RAM, destroying the previous contents of RAM. To use this routine:

- enter CALL 34. The terminal responds with "POSITION TAPE AND PRESS READ FROM TAPE".
- Press PLAY. Tape movement begins and the routine searches for the beginning of the next program. When the program is found an asterisk is printed for each line read from the recorder.

During this time the "DATA OUT" LED of the recorder and the "RECV" LED on the front of the module illuminate. When the recorder sends the last line of program, a final asterisk appears on the terminal indicating that the load operation is complete. Tape motion ceases and the BASIC prompt (>) appears.

If you are using a 1770–SA Recorder, press STOP.

Important: This routine loads the first program encountered and does not distinguish between labelled programs. See section 5.8.10 below titled, "Load Labeled Program from Data Recorder (1770–SB only) CALL 39" for an explanation of labelled programs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.6 Get Numeric Input Character from Peripheral Port – CALL 35

This routine gets the current character in the 255 character, peripheral port input buffer. It returns the decimal representation of the characters received as its output argument. The peripheral port 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 representation of that character. There is no input argument for this routine.

Example:

```
> 10 CALL 35
> 20 POP X
> 30 IF X =0 THEN GOTO 10
> 40 PRINT CHR(X)
```

Important: A 0 (null) is a valid character in some communications protocols. You should use CALL 36 to determine the actual number of characters in the buffer.

Important: Purge the buffer before storing data to ensure data validity.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.6 Get Numeric Input Character from Peripheral Port – CALL 35 (continued)

Example:

```
> 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

READY
>
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.7 Get the Number of Characters in the Peripheral Port Buffers – CALL 36

Use this routine to retrieve the number of characters in the chosen buffer as its output argument. You must PUSH which buffer is to be examined.

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

One POP is required to get the number of characters.

Example:

```
> 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 the Output Buffer is",X
> 50 END
```

5.8.8 Clear the Peripheral Ports Input or Output Buffer – CALL 37

This routine clears either the peripheral ports input or output buffer.

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

Example:

```
> 10 PUSH 0: REM clears the output buffer
> 20 CALL 37
> 30 END
```

5.8.9 Save Labeled Program to Data Recorder (1770-SB only) – CALL 38

Important: Maximum baud rate is 2400 bps for all data recorder CALL routines. The Peripheral Port is set up in these routines and does not need setting in BASIC. See section 4.4.3 for required cable connections. Tape verification failures result if you use a cable with different pin connections

This routine is identical to the CALL 32 routine except the program is assigned an ID number (0-255) before storing it on tape. This allows you to search for a specific program on a tape that contains many different programs. To do this:

- PUSH the ID number.
- enter CALL 38. From this point on operation is identical to the CALL 32 routine. See section 5.8.3 above titled, "Save Program to Data Recorder – Call 32" for an explanation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.9 Save Labeled Program to Data Recorder (1770-SB only) – CALL 38 (continued)

This routine has one input argument and no output arguments. If no ID number is pushed prior to CALL 38, an error occurs and the routine aborts without saving.

An unlabeled save (CALL 32) has an ID of 0.

Important: Do not press rewind between programs saved. If you press rewind only the last program is saved.

5.8.10 Load Labeled Program from Data Recorder (1770-SB only) – CALL 39

Important: Maximum baud rate is 2400 bps for all data recorder CALL routines. The Peripheral Port is set up in these routines and does not need setting in BASIC. The Peripheral Port parameters restore when the call is complete. See section 4.4.3 for required cable connections. Tape verification failures result if you use a cable with different pin connections.

This routine is identical to the CALL 34 routine except that labeled programs are organized according to an ID number (0-255) assigned to the stored program using a CALL 38. To use this routine:

- PUSH the ID number.
- Enter CALL 39. From this point on, operation is identical to the CALL 34 routine. See Section 5.8.4 above titled, "Verify Program with Data Recorder – CALL 33" for explanation.

This routine has one input argument and no output arguments. If no ID number is pushed prior to CALL 38, an error occurs and the routine aborts without saving.

While searching for the correct program, one asterisk prints for each program encountered. When the correct program is found one asterisk per line prints. You must press the READ FROM TAPE button twice on the SB Recorder at the end of each program encountered to continue the search. Press the READ FROM TAPE button when the DATA OUT light on the SB Recorder goes out. This indicates the end of file. If the BASIC Module starts loading from the middle of a file, the DATA OUT light is on but no asterisk is printed on the console. Follow the above procedure to continue the search.

An unlabeled save (CALL 32) has an ID of 0.

5.8.11 Print the Peripheral Port Output Buffer and Pointer – CALL 110

CALL 110 prints the complete buffer with addresses, front pointer and the number of characters in the buffer to the console. No PUSHes or POPs are needed. Use this information as a troubleshooting aid. It does not affect the contents of the buffer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.8.12 Print the Peripheral Port Input Buffer and Pointer – CALL 111

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

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

5.8.13 Reset the Peripheral Port to Default Settings – CALL 119

CALL 119 resets the peripheral port to the following default settings:

- 8 bits/character
- 1 stop bit
- No parity
- DCD off
- XON-XOFF off

No PUSHes or POPs are needed.

5.9 Wall Clock Support Calls

The battery backed wall time clock provides year, month, day of month, day of week, hours, minutes and seconds. The clock operates in 24 hour military time format. The support routines allow the setting of the clock and retrieval of clock values in numeric form.

The wall clock support routines use the argument stack to pass data between the BASIC program and the routines. Data is passed in both directions and consists of the actual clock data.

The wall clock or time of day clock is separate from the real time clock also provided on the module. The real time clock is accessed by CLOCK 1, CLOCK0, ONTIME and other statements, and has a resolution of 5 milliseconds. It should be used for all short time interval measurements because the greater resolution results in more accurate timing. The two clocks are not synchronized and comparison of times is not recommended. Also, the real time clock is not battery backed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.9.1 Setting the Wall Clock Time (Hour, Minute, Second) – CALL 40

Use this routine to set the following wall clock time functions:

- H = hours (0 to 23)
- M = minutes (0 to 59)
- S = seconds(0-59)

Example:

Program the wall clock for 1:35 pm (13:35 on a 24 hour clock).

```
> 10 H=13: M=35: S=00      :REM HOURS=13; MINUTES=35; SECONDS=00
> 20 PUSH H,M,S            :REM PUSH HOURS, MINUTES, SECONDS
> 30 CALL 40               :REM CALL THE ROUTINE TO SET THE WALL CLOCK TIME
```

5.9.2 Setting the Wall Clock Date (Day, Month, Year) – CALL 41

Use this routine to set the following wall clock functions:

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

Three values are PUSHed and none POPed.

Example:

Program the wall clock for the 16th day of June, 1985

```
> 10 D=16: M=06: Y=85      :REM DAY OF MONTH = 16, MONTH = 6, YEAR = 85
> 20 PUSH D,M,Y            :REM PUSH DAY OF MONTH, MONTH, YEAR
> 30 CALL 41               :REM CALL THE ROUTINE TO SET THE WALL CLOCK DATE
```

5.9.3 Set Wall Clock – Day of Week – CALL 42

CALL 42 sets the day of the week. Sunday is day 1. Saturday is day 7.

Example:

```
> PUSH 3 : CALL 42 REM DAY IS TUESDAY.
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.9.4 Date/Time Retrieve String – CALL 43

CALL 43 returns 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.

Example:

```
> 10 STRING 100,20
> 20 PUSH 1: CALL 43: REM put date/time in string 1
> 30 PRINT $(1)
> 40 END
```

5.9.5 Date Retrieve Numeric (Day, Month, Year) – CALL 44 (2)

CALL 44 returns 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 POPed in day, month and year order.

Example:

```
> 10 REM DATE RETRIEVE – NUMERIC EXAMPLE
> 20 CALL 44: REM INVOKE THE UTILITY ROUTINE
> 30 POP D,M1,Y : REM GET THE DATA FROM THE 30 STACK
> 40 PRINT "CURRENT DATE IS", Y,M1,D
> 50 END
> RUN

  CURRENT DATE IS 84 12 25

  READY
```

5.9.6 Time Retrieve String – CALL 45

CALL 45 returns 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.

Example:

```
> 10 STRING 100,20
> 20 PUSH 1, CALL 45: REM put time in string 1
> 30 PRINT $(1)
> 40 END
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.9.7 Time Retrieve Number – Call 46

The time of day is available in numeric form by executing a CALL 46 and POPing the three variables off of the argument stack on return. There are no input arguments. The time is POPed in hour, minute and second order.

Example:

```
> 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
> RUN
CURRENT TIME IS 13 44 54
READY
```

5.9.8 Retrieve Day of Week String – CALL 47

CALL 47 returns the current day of week as a three character string. PUSH the number of the string to receive the day of week. You must allocate a minimum of 3 characters/string. Strings returned are SUN, MON, TUE, WED, THUR, FRI, SAT.

Example:

```
> 10 PUSH 0: CALL 47
> 20 PRINT "TODAY IS",$(0)
```

5.9.9 Retrieve Day of Week Numeric – CALL 48

CALL 48 returns the current day of week on the argument stack as a number (i.e. Sunday = 1, Saturday = 7). This can be POPed into a variable.

Example:

```
> 10 REM DAY OF WEEK RETRIEVE – NUMERIC EX
> 20 CALL 44: REM INVOKE UTILITY TO GET D.O.W.
> 30 POP D
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.9.10 Date Retrieve String – CALL 52

CALL 52 returns 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.

Example:

```
> 10 STRING 100,20
> 20 PUSH 1: CALL 52: REM put date in string 1
> 30 PRINT $(1)
> 40 END
> RUN
30/JAN/87
```

5.10 Description of String Operators

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 deal with words instead of numbers. This allows you to refer to individuals by name instead of by number.

The BASIC Module contains one dimensioned string variable, $([expr]). The dimension of the string variable (the [expr] value) ranges from 0 to 254. This means that you can define and manipulate 255 different strings in the BASIC Module. Initially, no memory is allocated for strings. Memory is allocated by the STRING [expr], [expr] STATEMENT. See Section 5.4.31 for more information on this statement.

In the BASIC Module you can define strings with the LET statement, the INPUT statement and with the ASC () operator.

Example:

```
> 10 STRING 100,20
> 20 $(1)="THIS IS A STRING,"
> 30 INPUT "WHAT'S YOUR NAME? – ",$(2)
> 40 PRINT $(1),$(2)
> 50 END
> RUN
```

WHAT'S YOUR NAME? – FRED

THIS IS A STRING, FRED

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

Example:

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.1 The ASC Operator

Two operators in the BASIC Module can manipulate STRINGS. These operators are ASC() and CHR(). The ASC() operator returns the integer value of the ASCII character placed in the parentheses.

Example:

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

The decimal representation for the ASCII character "A" is 65. In addition, you can evaluate individual characters in a pre-defined ASCII string with the ASC() operator.

Example:

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

  THIS IS A STRING
  84
```

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.

Example:

```
> 5 STRING 1000,40
> 10 $(1)="ABCDEFGHIJKL"
> 20 FOR X=1 TO 12
> 30 PRINT ASC($(1),X),
> 40 NEXT X
> 50 END
> RUN

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.1 The ASC Operator (continued)

You can also use the ASC() operator to change individual characters in a defined string.

Example:

```
> 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)
> RUN

ABCDEFGHIJKL
KBCDEFGHIJKL
KCCDEFGHIJKL
```

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

Example:

```
> 5 STRING 1000,40
> 10 $(1)="SECRET" : REM SECRET IS THE PASSWORD
> 20 INPUT "WHAT'S THE PASSWORD – ",$(2)
> 30 FOR I=1 TO 6
> 40 IF ASC($(1),I)=ASC($(2),I) THEN NEXT I ELSE 70
> 50 PRINT "YOU GUESSED IT!"
> 60 END
> 70 PRINT "WRONG, TRY AGAIN" : GOTO 20
> RUN

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


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.2 The CHR Operator

The CHR() operator converts a numeric expression to an ASCII character.

Example:

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

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

Example:

```
> 5 STRING 1000,40
> 10 $(1)="The BASIC Module"
> 20 FOR I=1 TO 16 : PRINT CHR($(1),I), : NEXT I
> 30 PRINT: FOR I=16 TO 1 STEP -1
> 40 PRINT CHR($(1),I), : NEXT I
> RUN

The BASIC Module

eludoM CISAB ehT
```

In the above example, the expressions contained within the parenthesis, 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 a BAD SYNTAX ERROR. Use the ASC() operator to change a value in a string, or use the string support call routine – replace string in a string. See Section 5.10.3 below titled, "String Support Calls" for more information.

Important: Use the CHR() function only in a print statement.

5.10.2.1 Clearing the Screen on an Allen-Bradley Industrial Terminal

Refer to Appendix B at the rear of this manual. In "Column 1" under "DEC", you find that 12 equals ASCII FF (form feed). Type PRINT CHR(12) to clear the industrial terminal screen with a form feed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.2 The CHR Operator (continued)

5.10.2.2 Cursor Positioning on an Industrial Terminal

Allen-Bradley Industrial Terminal – Refer to the Industrial Terminal User's Manual (publication number 1770-6.5.3), Table 6-3. You can control cursor positioning by typing:

(CTRL)(P)(COLUMN#)(;)(ROW#)(A).

The ASCII equivalent of (CTRL)(P) is DLE. Using Appendix B, the DEC equivalent of ASCII DLE is 16.

```
PRINT CHR(16), "40;10ATEST"
```

This prints "TEST" at column 40 row 10 on your Allen-Bradley Industrial Terminal.

5.10.3 String Support Calls

To enhance the BASIC Module we have added several string manipulation routines that make programming easier. Strings in the BASIC Module are declared by the STRING statement. You must execute the STRING statement before you can access or call any strings or string routines. The STRING statement has two arguments or numbers that follow it. They are:

- the total amount of space to allocate to string storage and,
- the maximum size in characters of each string.

Since you terminate strings using a carriage return character, each string is given an extra byte of storage for its carriage return if it is the maximum length. You can determine the number of strings allowed by taking the first number and dividing it by one plus the second number. Note that you must use the strings consecutively starting with string 0 through the allowed number of strings. All strings are allocated the maximum number of characters regardless of the actual number used.

All of these routines use or modify strings as part of their operation. The mechanism for passing a string to the support routine is to PUSH its number or subscript onto the stack. The support routine can then get the string number from the module's argument stack, and locate it in string memory.

Many of these routines also require the length of a string as an input. This number must normally be inclusively between zero and the second number used in the last STRING statement which specifies the maximum size of a string. However, in all cases, if a string length argument of minus one (-1) is given, it is interpreted as the maximum allowable string length.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

It is important to note that since the carriage return character is the string terminator, you cannot use it within a string as one of its characters. If the high bit is set in a carriage return character (use 141 instead of 13 as the decimal value of the carriage return character) the BASIC Module does not recognize it as the end of string character and passes it to the output device. Most devices use a seven bit ASCII code, ignore the top bit and treat the 141 as a normal carriage return.

Important: All undefined characters of a string (i.e. characters following CR) are nulls. If data is input to a string using the ASC($(X),I) command, you must be sure the string is (CR) terminated properly. You can initialize the string or add a (CR) to terminate the string.

Example:

```
> 10 STRING 100,10
> 20 FOR I=1 TO 5
> 30 ASC($(0),I)=65
> 40 NEXT I
> 50 PRINT $(0)
> RUN
AAAAA
```

The string is stored in memory without a (CR) terminator. Be sure to insert a (CR) (0DH) into the last position of the string if using the above method.

Example:

```
> 10 STRING 100,10
> 20 FOR I=1 TO 5
> 30 ASC ($(0),I) = 65
> 40 NEXT I
> 45 ASC($(0),6)=0DH
> 50 PRINT $(0)
```

or

```
> 10 STRING 100,10
> 15 $(0))=“ -----”
> 20 FOR 1 TO 5
> 30 ASC ($(0),I) = 65
> 40 NEXT I
> 50 PRINT $(0)
```

The following three programs allow you to determine how much string space to allocate when you know two of three variables associated with the strings:

- number of characters in the longest string.
- number of string variables.
- amount of memory to allocate for strings.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

Example program 1:

```
> LIST
10      REM STRING ALLOCATION COMPUTATION KNOWING:
20      REM 1)# CHARACTERS IN LONGEST STRING 2)# OF STRING VARIABLES
30      PRINT : PRINT
40      INPUT "HOW MANY CHARACTERS IN YOUR LONGEST STRING"C
50      INPUT "HOW MANY STRING VARIABLES WILL YOU NEED"V
60      PRINT
70      N=(((C+1)*V)+1) : REM COMPUTE THE # OF BYTES OF MEMORY NEEDED
80      PRINT: PRINT
90      PRINT "YOU NEED TO ALLOCATE",N,"BYTES OF MEMORY FOR",V,"VARIABLES
100     PRINT "CONTAINING",C,"CHARACTERS EACH"
110     PRINT : PRINT
120     PRINT"                 STRING",N",",C
130     END
> RUN
HOW MANY CHARACTERS IN YOUR LONGEST STRING
?21
HOW MANY VARIABLES WILL YOU NEED
?15
YOU NEED TO ALLOCATE 331 BYTES OF MEMORY FOR 15 VARIABLES CONTAINING
21 CHARACTERS EACH
        STRING 331, 21
READY
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

Example program 2:

```
> LIST
10      REM STRING ALLOCATION COMPUTATION KNOWING:
20      REM 1)# CHARACTERS IN LONGEST STRING 2) AMOUNT OF STRING MEMORY
30      PRINT : PRINT
40      INPUT "HOW MANY CHARACTERS IN YOUR LONGEST STRING"C
50      INPUT "# OF BYTES OF MEMORY CAN YOU ALLOCATE FOR STRINGS"N
60      PRINT
70      V=INT((N-1)/(C+1)) : REM COMPUTE THE # OF POSSIBLE VARIABLES
80      N=(V*(C+1))+1 : REM COMPUTE HOW MUCH MEMORY IS ACTUALLY NEEDED
90      PRINT: PRINT
100     PRINT "YOU NEED TO ALLOCATE",N,"BYTES OF MEMORY FOR",V,"VARIABLES
110     PRINT "CONTAINING",C,"CHARACTERS EACH
120     PRINT: PRINT
130     PRINT"                  STRING",N,",",C
140     END

> RUN

HOW MANY CHARACTERS IN YOUR LONGEST STRING
?20
# OF BYTES OF MEMORY CAN YOU ALLOCATE FOR STRINGS
?500

YOU NEED TO ALLOCATE 485 BYTES OF MEMORY FOR 22 VARIABLES
CONTAINING 21 CHARACTERS EACH

        STRING 485 , 21
READY
>
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

Example program 3:

```
10      REM STRING ALLOCATION COMPUTATION KNOWING:
20      REM 1) AMOUNT OF STRING MEMORY 2) #OF STRING VARIABLES
30      PRINT: PRINT
40      INPUT "ENTER # OF BYTES OF MEMORY YOU CAN ALLOCATE FOR STRINGS"N
50      INPUT "HOW MANY STRING VARIABLES WILL YOU NEED"V
60      PRINT
70      C=INT(((N-1)/V)-1) : REM COMPUTE THE # OF CHARACTERS/STRING
80      N=(V*(C+1))+1 : REM COMPUTE THE # OF BYTES OF MEMORY NEEDED
90      PRINT: PRINT
100     PRINT "YOU NEED TO ALLOCATE",N,"BYTES OF MEMORY FOR", V,"VARIABLES"
110     PRINT "CONTAINING","C,CHARACTERS EACH"
120     PRINT: PRINT
130     PRINT"                STRING",N,",",C
140     END
> RUN

ENTER # OF BYTES OF MEMORY YOU CAN ALLOCATE FOR STRINGS 500
HOW MANY STRING VARIABLES WILL YOU NEED ? 15

YOU NEED TO ALLOCATE 496 BYTES OF MEMORY FOR 15 VARIABLES CONTAINING 32
CHARACTERS EACH

                                STRING 496 , 32

READY
>
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.1 String Repeat – CALL 60

This routine allows you 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 character to be repeated. No arguments are POPed. You cannot repeat more characters than the string's maximum length.

```
> 10 REM STRING REPEAT EXAMPLE PROGRAM
> 20 STRING 1000,50
> 30 $(1)="*"
> 40 PUSH 40 :REM THE NUMBER OF TIMES TO REPEAT CHARACTER
> 50 PUSH 1 :REM WHICH STRING CONTAINS CHARACTER
> 60 CALL 60
> 70 PRINT $(1)
> 80 END
> RUN
****************************************
READY
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.2 String Append (Concatenation) – CALL 61

This routine allows you to append one string to the end of another string. The 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)).

```
> 10 STRING 200,20
> 20 $(1)="How are"
> 30 $(2)="you?"
> 40 PRINT "BEFORE:",
> 50 PRINT "$1=",$(1)," $2=",$(2)
> 60 PUSH 2 :REM STRING NUMBER OF STRING TO BE APPENDED
> 70 PUSH 1 :REM BASE STRING NUMBER
> 80 CALL 61 :REM INVOKE STRING CONCATENATION ROUTINE
> 90 PRINT "AFTER:",
> 100 PRINT "$1=",$(1)," $2=",$(2)
> 110 END

> RUN

BEFORE:     $1=How are      $2=you?
AFTER:      $1=How are you? $2=you?

READY
```

5.10.3.3 Number to String Conversion – CALL 62

This routine converts 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.

- PUSH the value to be converted
- PUSH the number of the string to receive the value.

Example:

```
> 10 STRING 100,14
> 20 INPUT "ENTER A NUMBER TO CONVERT TO A STRING ",N > 30 PUSH N
> 40 PUSH 1: REM CONVERT NUMBER TO STRING 1
> 50 CALL 62: REM DO THE CONVERSION
> 60 PRINT $(1)
> 70 END
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.4 String to Number Conversion – CALL 63

This routine converts 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. Two bytes are POPed after the CALL:

- validity of the value
- actual value

If the string does not contain a legal value a 0 (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 value, then POP the value. If a string contains a number followed by an E followed by a letter or non-numeric character, it is assumed that no number was found since the letter is not a valid exponent (UAB701EA returns a zero in the first argument popped indicating that no valid number was in the string).

Example:

```
> 10 STRING 100,14
> 20 INPUT "ENTER A STRING TO CONVERT",$(1)
> 30 PUSH 1: REM CONVERT STRING 1
> 40 CALL 63: REM DO THE CONVERSION
> 50 POP V,N
> 60 IF V < > 0 THEN PRINT $(1)," ",N: GO TO 80
> 70 PRINT "INVALID OR NO VALUE FOUND" 80 END
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.5 Find a String in a String – CALL 64

This routine finds a string within a string. It locates the first occurrence (position) of this string. This call expects two input arguments . The first is the string to be found, the second is the string to be searched for a match. One return argument is required. If the number is not zero then a match was located at the position indicated by the value of the return argument. This routine is similar to the BASIC INSTR$(findstr$,str$). (Example: L=INSTR$($(1),$(2))

```
> 10    REM SAMPLE FIND STRING IN STRING ROUTINE
> 20    STRING 1000,20
> 30    $(1)="456"
> 40    $(2)="12345678"
> 50    PUSH 1 :REM STRING NUMBER OF STRING TO BE FOUND
> 60    PUSH 2 :REM BASE STRING NUMBER
> 70    CALL 64 :REM GET LOCATION OF FIRST CHARACTER
> 80    POP L
> 90    IF L=0 THEN PRINT "NOT FOUND"
> 100   IF L > 0 THEN PRINT "FOUND AT LOCATION ",L
> 110   END

> RUN

FOUND AT LOCATION 4

READY
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.6 Replace a String in a String – CALL 65

This routine replaces a string within a string. Three arguments are expected. The first argument is the string number of the string which replaces the string identified by the second argument string number. The third argument is the base string's string number. There are no return arguments.

```
> 10   REM SAMPLE OF REPLACE STRING IN STRING
> 20   STRING 1000,20
> 30   $(0)="RED-LINES"
> 40   $(1)="RED"
> 50   $(2)="BLUE"  > 60 PRINT "BEFORE: $0=",$(0)
> 70   PUSH 2 :REM STRING NUMBER OF THE STRING TO REPLACE WITH
> 80   PUSH 1 :REM STRING NUMBER OF THE STRING TO BE REPLACED
> 90   PUSH 0 :REM BASE STRING NUMBER
> 100  CALL 65 :REM INVOKE REPLACE STRING IN STRING ROUTINE
> 110  PRINT "AFTER: $0=",$(0)
> 120  END
RUN
BEFORE: $=RED-LINES
AFTER: $0=BLUE-LINES
READY
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.7 Insert String in a String – CALL 66

This routine inserts 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 string number of the base string. This routine has no return arguments.

```
> 10   REM SAMPLE ROUTINE TO INSERT A STRING IN A STRING
> 20   STRING 500,15
> 30   $(0)="1234590"
> 40   $(1)="67890"
> 50   PRINT "BEFORE: 0$=",$(0)
> 60   PUSH 6 :REM POSITION TO START THE INSERT
> 70   PUSH 1 :REM STRING NUMBER OF THE STRING TO BE INSERTED
> 80   PUSH 0 :REM BASE STRING NUMBER
> 90   CALL 66 :REM INVOKE INSERT A STRING IN A STRING
> 91   REM :REM ROUTINE
> 100  PRINT "AFTER: 0$=",$(0)
> 110  END

> RUN

BEFORE: 0$=1234590
AFTER: 0$=1234567890
READY
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.8 Delete String from a String – CALL 67

This routine deletes a string from within another string. The call expects two arguments. The first argument is the base string number. The second is the string number of the string to be deleted from the base string. This routine has no return arguments.

Important: This routine deletes only the first occurrence of the string.

```
> 10   REM ROUTINE TO DELETE A STRING IN A STRING
> 20   STRING 200,14
> 30   $(1)="123456789012"
> 40   $(2)="12"
> 50   PRINT "BEFORE: $1=",$(1)
> 60   PUSH 1 :REM BASE STRING NUMBER
> 70   PUSH 2 :REM STRING NUMBER OF THE STRING TO BE DELETED
> 80   CALL 67 :REM INVOKE STRING DELETE ROUTINE
> 90   PRINT "AFTER: $1=",$(1)
> 100  END

> RUN

BEFORE: $1=123456789012
AFTER: $1=3456789012

READY
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.10.3 String Support Calls (continued)

5.10.3.9 Determine Length of a String – CALL 68

This routine determines 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($1)).

```
> 10    REM SAMPLE OF STRING LENGTH
> 20    STRING 100,10
> 30    $(1)="1234567"
> 40    PUSH 1 :REM BASE STRING
> 50    CALL 68 :REM INVOKE STRING LENGTH ROUTINE
> 60    POP L :REM GET LENGTH OF BASE STRING
> 70    PRINT "THE LENGTH OF ",$(1)," IS ",L
> 80    END

> RUN

THE LENGTH OF 1234567 IS 7

READY
```

5.11 Memory Support Calls

The following sections list and describe the memory support calls you can use with the BASIC Module. All strings, arrays and variables are shared.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.11.1 ROM to RAM Program Transfer – CALL 70

This routine shifts program execution from a running ROM program to the beginning of the RAM program. No arguments are PUSHed or POPed.

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

Example:

ROM #5

```
> 10 REM SAMPLE ROM PROG FOR CALL 70
> 20 PRINT "NOW EXECUTING ROM #5"
> 30 CALL 70 :REM GO EXECUTE RAM
> 40 END
```

RAM

```
> 10 REM SAMPLE RAM PROGRAM FOR CALL 70
> 20 PRINT "NOW EXECUTING RAM"
> 30 END

> RUN

NOW EXECUTING ROM #5
NOW EXECUTING RAM
```

5.11.2 ROM/RAM to ROM Program Transfer – CALL 71

This routine transfers from a running ROM or RAM program to the beginning of any available ROM program. One argument is PUSHed (which ROM program). None are POPed. 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.

Example:

```
> 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
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.11.2 ROM/RAM to ROM Program Transfer – CALL 71 (continued)

The user is now executing ROM 4 if it exists. If the ROM routine requested does not exist the result is:

```
PROGRAM NOT FOUND.
READY
>
```

5.11.3 RAM/ROM Return – CALL 72

This routine allows you 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 POPed. This routine works one layer deep. You may go back to the last CALLing program's next line.

Important: There must be a next line in the ROM or RAM routine, otherwise unpredictable events could occur which 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.

Example:

```
> ROM #1

> 10    REM SAMPLE PROG FOR CALL 72
> 20    PRINT "NOW EXECUTING ROM #1
> 30    PUSH 3
> 40    CALL 71 :REM EXECUTE ROM #3 THEN RETURN
> 50    PRINT "EXECUTING ROM #1 AGAIN"
> 60    END
```

ROM #3

```
> 10    REM THIS LINE WONT BE EXECUTED
> 20    PRINT "NOW EXECUTING ROM #3"
> 30    CALL 72
> 40    END
```

With ROM #1 selected:

```
> RUN

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

READY
>
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.11.4 Battery-backed RAM Disable – CALL 73

CALL 73 disables the battery-backed RAM, prints "Battery Backup Disabled" when executed and allows a purging reset. The next power loss destroys the contents of RAM. When power is reapplied, RAM is cleared and battery back-up is reenabled automatically.

5.11.5 Battery-backed RAM Enable – CALL 74

CALL 74 enables the battery-backed RAM and prints "Battery Backup Enabled" when executed. It is enabled on power-up and remains enabled until you execute a CALL 73 or until the battery fails.

5.11.6 Protected Variable Storage – CALL 77

Important: Change MTOP from command mode only to ensure proper operation

CALL 77 reserves 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 to be stored from MTOP reducing available RAM memory. This value is PUSHed onto the stack as the new MTOP address. All appropriate variable pointers are reconfigured. 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.

Example: For saving 2 variables.

```
> PRINT MTOP
14335
> PRINT MTOP-12
14323
> PUSH 14323:REM NEW MTOP ADDRESS
> CALL 77

> 10 K = 678 * PI
> 15 L =520 PUSH K
> 30 ST @ 14335:REM STORE K IN PROTECTED AREA
> 40 PUSH L
> 50 ST @ 14329
> 55 REM TO RETRIEVE PROTECTED VARIABLES
> 60 LD@ 14335:REM REMOVE K FROM PROTECTED AREA
> 70 POP K
> 80 LD@ 14329
> 90 POP L
> 100 REM USE LD@ AFTER POWER LOSS AND BATTERY BACK-UP IS USED
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.11.6 Protected Variable Storage – CALL 77 (continued)

Example:

For using protected variable storage area.

A. Scalar Variables

Push all variables in one line:

```
> PRINT MTOP
14335
> PRINT MTOP-24
14311
> PUSH 14311:REM NEW MTOP ADDRESS
CALL 77

90 M1=14335 : REM BEGIN STORING HERE
100 PUSH A, B, C, D
```

Use the ST @ and LD @ commands in a DO loop:

```
> 200 DO
> 210 ST @ M1
> 220 M1=M1-6 : REM EACH VARIABLE =6 BYTES
> 230 UNTIL M1=MTOP: REM YOU DEFINED THE NEW MTOP W/CALL 77
> 290 M1=14335
> 300 DO
> 310 LD @ M1
> 320 M1 = M1-6
> 330 UNTIL M1 = MTOP
> 360 POP A, B, C, D
> 370 PRINT A, B, C, D
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.11.6 Protected Variable Storage – CALL 77 (continued)

B. Arrays

Use an array to set up the data

```
> PRINT MTOP 14335
> PRINT MTOP-24 14311
> PUSH 14311:REM NEW MTOP ADDRESS
CALL 77
```

```
100 DIM A(4)
110 DATA 10, 20, 30, 40
120 FOR I=1 TO 4: READ A(I) : NEXT I
```

Use ST@ and LD @ commands in a DO loop:

```
> 190 M1=14335
> 200 DO
> 210 PUSH A(I) : ST @ M1
> 220 I = I+1 : M1 = M1-6
> 230 UNTIL I = 4
> 290 M1=14335
> 300 DO
> 310 LD @ M1 : POP A(I)
> 320 PRINT A(I)
> 330 I = I+1 : M1 = M1-6
> 340 UNTIL I 10
```

5.12 Miscellaneous Calls

The following sections list and describe the miscellaneous calls you can use with the BASIC Module.

5.12.1 Program Port Baud Rate – CALL 78

CALL 78 allows you 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 19.2 K baud. PUSH the desired baud rate and CALL 78. The program port remains at this baud rate unless CALL 73 is invoked or the battery is removed and power is cycled. If this happens the baud rate defaults to 1200 baud.

Example:

```
> 10 PUSH 4800
> 20 CALL 78
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.12.2 Blink the Active LED by Default – CALL 79

The Active LED is on constantly during program execution or command mode. When you issue a CALL 79 the Active LED remains on while a program executes and blinks every second when the module is in the command mode. Issue CALL 79 again to cancel the blinking LED.

5.12.3 Check Battery Condition – CALL 80

CALL 80 checks the module's battery condition. If a 0 is POPed after a CALL 80, battery is okay. If a 1 is POPed a low battery condition exists.

Example:

```
> 10    CALL 80
> 20    POP C
> 30    IF C < > 0 THEN PRINT "BATTERY LOW"
> 40    END
```

5.12.4 User PROM Check and Description – CALL 81

Use CALL 81 before burning a program into PROM memory. This routine:

- determines the number of PROM programs.
- determines the number of bytes left in PROM.
- determines the number of bytes in the RAM program.
- prints a message telling if enough space is available in PROM for the RAM program.
- checks PROM validity if no program is found.
- prints a good or bad PROM message. A bad PROM message with an address of 00xx indicates an incomplete program.
- cannot detect a defective PROM.

No PUSHes or POPs are needed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.12.5 Reset Print Head Pointer – CALL 99

You can use CALL 99 when printing out wide forms to reset the internal print head character counter and prevent the automatic CR/LF at character 79. You must keep track of the characters in each line.

You can solve this problem in revision A or B modules using DBY(1 6H) = 0

Example:

```
> 10    REM THIS PRINTS TIME BEYOND 80TH COLUMN
> 20    PRINT TAB (79),
> 30    CALL 99
> 40    PRINT TAB (41), "TIME —",
> 50    PRINT H, ":", M ,":",S
> 60    END
```

5.12.6 Print the Argument Stack – CALL 109

CALL 109 prints 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, or pointer to the stack.

5.12.7 Print the Peripheral Port Output Buffer and Pointer – CALL 110

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

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

5.12.8 Print the Peripheral Port Input Buffer and Pointer – CALL 111

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

5.12.9 Reset the Peripheral Port to Default Settings – CALL 119

CALL 119 resets the peripheral port to the following default settings:

- 8 bits/character
- 1 stop bit
- No parity
- DCD off
- XON-XOFF off

No PUSHes or POPs are needed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Programming

6.1 Chapter Objectives

After reading this chapter you should be able to:

- program your BASIC Module for use with a programmable controller.
- use block-transfer to communicate with a programmable controller.

This chapter shows the BASIC programming and ladder logic needed for use with your processor. It also gives you sample programs.

6.2 Block-Transfer with the BASIC Module

Your BASIC Module communicates with any processor that has block-transfer capability. Your ladder logic program and BASIC program work together to enable proper communications between the module and processor.

The BASIC Module is a bi-directional block-transfer module. Bi-directional means that the module performs both read and write block-transfer operations. You transfer data (1 to 64, 16 bit words) from your module to the processor's data table with a block-transfer-read (BTR) instruction. You transfer data (1 to 64, 16 bit words) to your module from the processor's data table with a block-transfer-write (BTW) instruction.

Important: The module's read and write instruction enable bits must not set at the same time. Your program must toggle requests for the read and write instructions as shown in the sample programs.

6.2.1 Block-Transfer-Write and Block-Transfer-Read Buffers

The BASIC Module processor maintains a block-transfer-write (BTW) buffer containing the values of the last BTW sent by the host processor. You initialize this buffer using a CALL 4. You transfer data to the BASIC Module's buffer using CALL 6 or CALL 3. This buffer freezes and does not change when CALL 6 or CALL 3 completes. This double buffering and freezing ensures that the data does not change during processing by the BASIC Module. The BTW buffer remains unchanged and is accessed repeatedly by any of the data access routines provided. An additional CALL 6 or CALL 3 changes the BTW buffer contents.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP