GOT0 Control is given to the specified line number.

Syntax : GOT0 expr.

Expr. is a numerical expression, that results in a line number (computed GOTO).

If expr. results in a line number of an executable statement, that statement and those following are executed.

The GOT0 statement can be used in the prompt mode to re-enter a program at a desired point. The variables will not be cleared in that case.

EXAMPLE

See GOSUB example.

HR(x) Read or write a Holding Register over the I/O bus

Syntax : HR(x) = y write to HR 'x'
y = HR(x) read HR 'x' from PLC

When the BASIC interpreter encounters the HR(x) statement, the executive firmware is preprogrammed to interrogate the PLC via the I/O bus. A special handshaking program in the PLC receives this special request and puts the answer back on the I/O bus. The -1799 receives the data (in the case of a read) or an acknowledgment (in the case of a write). This function will read and write one HR per scan. If the handshake ladder program is not loaded into the PLC, the BASIC program will hang as it executes any statement that contains the reference HR(x).

Warning

Use of the HR(x) function in industrial applications is not recommended with firmware V1.5 or earlier. A running program may lock up on the HR(x) statement if communications with the host CPU via the I/O bus is corrupted. The HR(x) does not attempt a retry. All tasks programmed in BASIC will halt.

Communication via the OR(x), IR(x), IG(x) or OG(x) functions is the preferred method. Alternatively, the BREAD and BWRIT functions may be used to reliably extract data from, or to place data in, a PLC.

IF..THEN

Makes a decision regarding program flow based on the result of a relationship between two expressions.

Syntax : IF condition [THEN] statement.

Condition is a relationship test between two expressions.

Valid relationship characters are:

>	greater than
<	less than
=	equal
>=	greater than or equal
<=	less than or equal
<>	not equal
><	not equal.

The above relationships are applicable for all numerical expressions. The only valid relationship test for string expressions is = (equal).

EXAMPLES

```
IF X=Y GOT0 10
IF X>Y IF A>B THEN C=0
IF OG(1).$8000<> = 0 THEN PRINT "Bit 16 SET"
IF MID$(A$,3,2)=B$ GOTO 100
```

IG(x)

Reads or writes an Input Group.

Syntax : IG(x) = y write Input Group 'x'
y = IG(x) read Input Group 'x'

INPUT Receives characters from a port during program execution.

Syntax : INPUT ["prompt";]variable[,variable]

Prompt is a string constant which will be used to prompt for the desired input.
If prompt is followed by ; a question mark will be printed.
If prompt is followed by , the question mark is suppressed.

The data that is entered, is assigned to the variable(s the variable list. Furthermore, this data must meet the format, required by the variables, otherwise an error may occur.

If the variable list defines more input than is entered, an additional question mark is printed.
Numbers inputted must be separated by a comma.
A string variable can be up to 80 characters long.
Entry of numbers out of the range +32767 or -32768 causes an error.
If the first character of an inputted number is alphabetic, the system prompts with RE-ENTER.
If the first character of an inputted number is $, hexadecimal input is presumed.

An INPUT only stops execution of its own task. All other tasks continue to operate.

EXAMPLES

```
INPUT "ENTER PARAMETER";A
INPUT 'ENTER 2 NUMBERS",A,B
INPUT A$
INPUT 'SET STRING LENGTH";A$(0)
INPUT IR(9)
```

INSTR Searches for the first occurrence of B$ in A$ and returns the value of the position, at which the match is found.

Syntax : V = INSTR(A$, B$)

A$ and B$ may be string variables, expressions or constants. If no match is found, the result V=O. In all other cases, V results in a value between 1 and 80.

EXAMPLES

```
PRINT INSTR("ABCDEFG","DE")          (will print 4)
PRINT INSTR("ONE TWO THREE', B$)
A = INSTR(A$,"STOP")
```

IR(x) Read or write an Input Register

Syntax : IR(x) = y write to Input Register 'x'
y = IR(x) read Input Register 'x'

EXAMPLE

```
100  A = IR(1)      :REM Read IR0001
110  IR(2)=7        :REM Change IR0002 to 7
```

LED2 Turns LED status light 2 (lower) on (briefly) or off

Syntax : LED2 ON turns LED 2 on briefly (200 mS)

or LED2 OFF turns LED 2 off

LED Turns the module status led on or off.

Syntax : LEDON

or LEDOFF

LEDOFF turns the led, near to PORT C, off.
LEDON turns the led, near to PORT C, on.

The other LED, near to PORT D, can be turned on and off by executing these statements:

```
POKE($C60E,$10)  (on)
POKE ($C60F,$10) (off)
```

LEDON is similar to POKE ($C60E,8).
LEDOFF is similar to POKE ($C60F,8).

LET Assigns the value of an expression to a variable.

Syntax : LET variable= expression.

Variable is a valid variable name (see chapter 5). Expression must match the format of the variable, otherwise an error occurs, The word LET is optional.

String expressions may be additions of string expressions.

The arithmetic operators in expression are:

*	multiply
/	divide
	logical and
^	logical or (hex $5E)
!	logical exclusive or
+	add
	subtract

The three logical operators have the same priority as * and /. Divide by zero, over- and underflow at multiply and divide result in an error.

EXAMPLES

```
LET A$ = 'ABCDEFG' + B$ + MID$ (C$,3,4)
X=Y+B*(C-5)
HR(8) = IR(5).$7FFF
```

LINE INPUT Reads an entire line (up to 80 char.) from the input buffer into a string variable, ignoring the delimiter (comma).

Syntax : LINE INPUT["PROMPT";]variable[,variable]

LINE INPUT does exactly the same as INPUT, except when a string variable is part of the input list. At that time, data is assigned to that string variable until a carriage return is entered.

LIST Lists the program lines, specified by the line specification.

Syntax : LIST n - m
LIST n
LIST

n is the first line, to be listed.
m is the last line, to be listed.

LLIST Lists the program lines on the printer (port B).

Syntax : LLIST n - m
LLIST n
LLIST

n is the first line, to be printed.
m is the last line, to be printed.

Baud and data format for the printer are set up using the PORT command.

LOAD Reads a program from a tape loader, connected to port B.

Syntax : LOAD

A step by step procedure for loading and saving programs is given in appendii C.

LOC

Returns the number of characters in the associated input buffer.

Syntax : variable = LOC

LOC ranges from 0 to 127.

To get the number of characters waiting to be read from port C, LOC will only give that number if the I/O is redirected to that port:

PORT C : X=LOC:PORT A:PRINT X

LOC is useful if you want to test for data, before using the GET statement.

EXAMPLE

```
10    IF LOC<10 GOT0 1000        : REM NO INPUT YET
20    GET A$,10                  : REM READ 10 CHAR.
```

MBRCV Read a block of registers from a Modbus slave

Syntax : MBRCV address,HR,count,buffer,control1,control2,error

address		Modbus slave address (l-255)
HR		first address to read (i.e. HR0001 = address 0000)
count		number of registers to read (l-64)
control1	-	High byte: response timeout (l-255 20 mS ticks) Low byte : TxD delay after CTS (l-255 20 mS ticks)
control2	-	High byte: not used Low byte : maximum retry attempts (0 is 1 retry)
error		error code returned (refer to "Modbus PLC link error status' page 116 for listing)

This function uses Modbus function code 3 (register read). Modbus protocol only defines addresses 1-247. If this module is included in a network that includes non-Westinghouse equipment, no drop should be configured with addresses 248-255.

The MBRCV function reads PLC addresses directly. For this reason, a relationship between PLC memory and HR references is important. The relationship is:

HR reference = PLC memory + 1

or,

PLC memory = HR reference - 1

Warning

The MBRCV function will affect the RTS state of a PRINT, BREAD or BWRIT function under certain circumstances. When using the MBRCV or MBXMT function, it is recommended a program watchdog be monitored. If the watchdog times out, a software restart can be enabled. See Appendix K, Application Note 2 (page 99) for more information.

EXAMPLE

```
100  MBRCV 1,0,125,A(1),$FF00,2,E :REM Read HR1-HR125 from drop 1 and place
                                    in A(1) - A(125)
```

EXAMPLE

Read a block of data from a Modbus slave, then write this same data into a Westinghouse PLC.

```
100  PORT C 9600,N,8,1              :REM 9600 baud, 8 data, no parity, 1 stop bit
110  MBRCV 1,0,10,A(1),$FF00,2,E :REM Read HR1 through HR10
120  PORT B 9600,O,8,2              :REM 9600 baud, 8 data, odd parity, 2 stop bits
130  BWRIT 1,10,A(1),E              :REM transfer this data to PLC
```

NOTE: CTS on the selected RS-232 port must be high for MBRCV to operate. For information on Modbus slave capability refer to Appendix L (page 103).

MBXMT Write to a block of registers in a Modbus slave

Syntax : MBXMT address,HR,count,buffer,control1,control2,error

address	-	Modbus slave address (O-255)
HR		first address to read (HR0001 = PLC address 0000)
count		number of registers to read (l-64)
control1	-	High byte: response timeout (i-255 20 mS ticks)
		Low byte : TxD delay after CTS (l-255 20 mS ticks)
control2	-	High byte: not used
		Low byte : maximum retry attempts
error		error code returned (refer to "Modbus PLC link error status' page 116 for listing)

This function uses Modbus function code 16 (multiple register write). Address 0 is a broadcast message to all drops. Addresses 248-255 are reserved by certain Modbus drops not manufactured by Westinghouse. It is recommended that networks including non-Westinghouse components restrict themselves to addresses l-247.

Warning

The MBXMT function will affect the RTS state of a PRINT, BREAD or BWRIT function under certain circumstances. When using the MBRCV or MBXMT function, it is recommended a program watchdog be monitored. If the watchdog times out, a software restart can be enabled. See Appendix K, Application Note 2 (page 99) for more information.

NOTE: CTS on the selected RS-232 port must be high for MBRCV to operate. For information on Modbus slave capability refer to Appendix L (page 103).

MID$ Returns a part of a string variable (into another string variable).

Syntax : stringvarl = MID$ (stringvar2,n,m)

String var is a string variable A$...B$.
n is the position in stringvar2 that indicates where the new string will start.
m is the number of characters, that will be taken from stringvar2.

EXAMPLE

```
10  A$="ABCDEFG"
20  B$=MID$(A$,2,3)
30  PRINT B$        :REM BCD will be printed.
```

NEW Erases the entire program and clears all data.

Syntax : NEW

There is no way that a program can be recovered from a NEW command.

All ports are initialized to their default set-up, program and data memory is cleared.

NEXT Indicates the end of a program loop.

Syntax : NEXT var

Var is a valid name for a numeric variable.

See also FOR..NEXT.

FOR..NEXT loops can be nested up to a depth of 8.

OG(x) Reads an Output Group

Syntax : y = OG(x) reads Output Group 'x'

Output Groups can be read only. Attempting to write to an Output Register will indeed place data into the 1799 module Dual Port RAM module, but this data is not written to the I/O bus. Furthermore, during the next I/O update cycle of the CPU, this data in Dual Port RAM will be overwritten by the most current OG value held in the CPU output image table.

OPEN Sets port B open for input, and suppresses echoing of inputted characters, for the purpose of reading data from a tape recorder.

Syntax : OPEN

Detailed information about saving and loading data to/from tape can be found in appendix D.

OR(x) Reads an Output Register

Syntax : y = OR(x) reads Output Register 'x'

Rules similar to those for the OG(x) function pertain to the OR(x) function.

PAUSE Holds program execution until a carriage return is entered. When program execution is halted, the word PAUSE followed by the line number, is printed.

Syntax : PAUSE

EXAMPLE

```
10    FOR A=1 TO IO: PRINT "EXECUTING":NEXT A
20    PRINT 'HIT RETURN TO CONTINUE';: PAUSE
30    GOT0 10
```

PEEK Reads a byte from the indicated memory location.

Syntax : var = PEEK (address)

Address is a numerical expression, that results in a value 0 - 65535 ($00-$FFFF), and defines the memory location to be read.

For an overview of system peeks and pokes and useful memory locations, see appendix B.

POKE Writes a byte into the indicated memory location.

Syntax : POKE (address,byte)

Address is a numerical expression, that results in a value 0 - 65535 ($OO-$FFFF), and defines the memory location to be written to.

Byte is a numerical expression, that results in a value 0 - 255 ($00-$FF), and defines the value to be written.

See also PEEK

EXAMPLE

```
10      REM FLASH SECOND LED
20      POKE ($C60E,$10):SUSPEND 50
30      POKE ($C60F,$10):SUSPEND 50
40      GOT0 20
```

The previous example program could have used the statements LED2ON and LED2OFF.

PORT Sets data format and baud of a communication channel, and redirects input and output to that channel.

Syntax : PORT X [baud,par,bits,stop bits]

Baud	defines the baud rate (300,600,1200,2400,4800,9600 or 19200).
Par	defines the parity N for no parity, 0 for odd parity, E for even parity.
Bits	defines the number of data bits (7 or 8).
Stopbits	defines the number of stopbits (1 or 2)
X	defines the channel (A,B,C or D).

Up to 38,400 BPS can be selected. Refer to Appendix B.

The PORT statement also redirects input and output. The data format and baud specifications may be omitted.

A special format of the PORT statement is:

PORT X **S**
and PORTX E

PORTX **S**	suppresses echoing of inputted characters (silent mode).
PORT X E	enables echo.

This command is useful when interfacing devices, that do not allow for full-duplex operation.

EXAMPLE

```
10    A= 19200
20    PORT B A,N,8,2:REM SET DATA FORMAT
30    PORT B S
40    INPUT"ENTERPASSWORD";A$
50    IF A!§ = 'PWD'GOTO 100
60    GOT0 40
100   END
```

PRHEX Prints numerical expressions in hexadecimal format. PRHEX is identical to PRINT, except for the format that it prints numbers.

Syntax : PRHEX [list of expressions] [;]

If the delimiter between two expressions is a comma (,) tabulation (zone spacing) is done.

If the delimiter between two expressions is a semicolon (;), the zone spacing is suppressed.

A TAB (expr.) statement, used within a print statement, positions the cursor at a location specified by (expr.). If the cursor is beyond the specified location, printing starts at the current location.

If an expression is preceded by the # sign, the expression truncated to the lower 8 bits is printed as one character (for instance PRHEX #7 sends a bell character).

A semicolon at the end of a print statement suppresses carriage return and linefeed.

The PRHEX command will not send any data out the serial port unless the RTS lead for that port is raised HIGH by the 1799 and the CTS lead is raised high by the device,

POKE ($C70E,2)	Raise RTS port A
POKE ($C70E,1)	Raise RTS port B
POKE ($C60E,1)	Raise RTS port C

Note

If the RTS lead for the chosen port is not raised high, this function will not send data out of that serial port. The program will not, however, hang on that statement.

This can be a troublesome bug unless a protocol analyzer (or terminal) is connected to the serial port and the data that is transmitted is analyzed.

NOTE: CTS on the selected RS-232 port must be high for PRHEX to operate.

EXAMPLES

PRHEX A	prints the value of A in hexadecimal format. If the value of A is lower than 256, 2 characters are printed (00-FF), otherwise, 4 characters are printed (0100-FFFF).
PRHEX 'STRING"	prints the string between quotes.

NOTE: Execution continues to subsequent program statements after a PRHEX statement as soon as all but the last two bytes of the message have been printed. Care must be taken to insure that no other commands (PORT, POKE, etc.) manipulate that serial port before those characters have been sent out the port.

Questionable 100 PORT C:PRHEX 'This is a test":PORT B

Proper 100 PORT C:PRHEX "This is a test":SUSPEND 1 :PORT B

PRINT Prints numerical expressions in decimal format. PRINT is identical to PRHEX except for the format that it prints numbers.

Syntax : PRINT [list of expressions] [;]

If the delimiter between two expressions is a comma (,) tabulation (zone spacing) is done.

If the delimiter between **two expressions** is a semicolon (;), the zone spacing is suppressed.

A TAB (*expr.*) statement, used within a print statement, positions the cursor at a location specified by (expr.). If the cursor is beyond the specified location, printing starts at the current location.

If an expression is preceded by the **#** sign, the expression truncated to the lower 8 bits is printed (for instance PRINT **#7** rings a bell).

A semicolon at the end of a print statement suppresses carriage return and linefeed.

The PRINT command will not send any data out the serial port unless the RTS lead for that port is raised HIGH by the 1799 and the CTS lead is raised high by the device,

POKE (**$C70E,2**)	Raise RTS port A
POKE (**$C70E**, 1)	Raise RTS port **B**
POKE (**$C60E,1**)	Raise RTS port C

Note

If the RTS lead for the chosen port is not raised high, this function will not send data out of that serial port. The program will not, however, hang on that statement.

This can be a troublesome bug unless a protocol analyzer (or terminal) is connected to the serial port and the data that is transmitted is analyzed.

NOTE: CTS on the selected RS-232 port must be high for PRINT to operate.

EXAMPLES

PRINT	prints a return and linefeed.
PRINT A,B	prints two values in decimal format, with zone spacing.
PRINT **A$,B$,TAB(30);X**	
PRINT **#0;#1;#2;#A;**	print raw binary 0, **1,** 2 and the value in A
PRINT **#$FFFF;**	print raw binary value $FFFF

NOTE: Execution continues to subsequent program statements after a PRINT statement as soon as all but the last two bytes of the message have been printed. Care must be taken to insure that no other commands (PORT, POKE, etc.) manipulate that serial port before those characters have been sent out the port.

Questionable	100 PORT **C:PRINT** 'This is a **test":PORT** B
Proper	100 PORT **C:PRINT** "This is a **test":SUSPEND** I:PORT B

PROGRAM Selects a BASIC program from the optional EPROM and places it into memory.

Syntax : PROGRAM [n]

n is optional and defines the n.th additional program in EPROM.

The EPROM can be programmed on a universal PROM programmer that supports the MOTOROLA Exorciser down-load format.

The prom is located on the processor board, see appendix I (see page 91).

If the program specified does not exist or can not be found, the module terminates execution and prints an error message out the current serial port.

See also DWLOAD (page 39) and UPLOAD (page 65) commands,

REM Indicates that the remaining text of the line has to be regarded as explanatory remarks,

Syntax : REM remark

The BASIC interpreter skips control to the next line number, if a REM statement is encountered.

EXAMPLE

```
5     DIM B(100)
10    REM INITIALIZATION
20    FOR A=1 TO 100
30    B(A) = A*2:NEXT A
40    PRINT 'ENTER KEY":REM INPUT TESTED PERIODICALLY
50    IF LOC=O GOT0 100:REM NO INPUT YET
60    GET A: PRINT #A,A:GOTO 40
100   PRINT "?":SUSPEND 100:GOTO 50
```

RETURN Marks the end of a subroutine and causes the interpreter to branch back to the statement, following the most recent GOSUB statement.

Syntax : RETURN

Executing a RETURN without a corresponding GOSUB will result in the module terminating execution and printing an error message out the current serial port.

EXAMPLE

```
5     DIM B(lOO)
10    GOSUB 100
20    |
      |
      |
100   REM SUBROUTINE INIT ARRAY B
110   FOR A=1 TO 100:B(A)=A:NEXT A
120   RETURN
```

RUN Executes the program.
All variables are cleared.

Syntax : RUN

SAVE Saves a program in memory to a tape loader, connected to port B.

Syntax : SAVE

A step by step procedure for loading and saving programs is given in appendix C (page 77).

SETDATE Used to change the HOURS, MINUTES and/or SECONDS value of the real time clock.

Syntax : SETDATE (see Appendix H, page 89, for more information)

SETTIME Used to change the DAY, MONTH, DATE and YEAR value of the real time clock.

Syntax : SETTIME (see Appendix H, page 89, for more information)

SIZE Prints the number of bytes used by the program, and the number of free bytes.

Syntax : SIZE

Array storage is not included, until the program has been run.

EXAMPLE

```
10      SIZE
20      DIM A(100)
30      SIZE
```

PEEK($856) and PEEK($857) can be read to calculate the size of the BASIC program code alone.

SUSPEND

Suspends execution of a task for a specified period.

Syntax : SUSPEND expression

Expression defines the number of system ticks of 20 mS, that the task is suspended.

SUSPEND is a useful command for replacing FOR..NEXT delay loops, because it does not consume any processor time. Consequently, other tasks benefit from the gained processor time.

EXAMPLE

```
90   REM IF TASK NOT REQUESTED, SUSPEND IT
100  IF A=0 SUSPEND 100:GOTO 100
110  REM PERFORM TASK
120  A=0:REM RESET REQUEST
130  PRINT 'TASK IS ACTIVE"
140  FOR C=1 TO 1000:NEXT C:PRINT 'TASK DONE'
150  GOT0 100
```

SYNC

Synchronizes with a Westinghouse PC 700, 900, 1100, 1200 or 1250 using the 6 byte point to point protocol.

Syntax : SYNC ticks, total, error

ticks - amount of time to wait between each null (binary 0) character sent. Each tick is 20 mS.

total - maximum number of nulls to send before quitting with an error code. If this function is unable to synchronize (receive a response) from the PLC after this many null characters have been sent, the function will abort and control will be passed to the next program statement.

error - 0 - successful synchronization

Example

```
90   BREAD 1,1,A,E                    :REM Read HR1 from PLC
100  IF E <> 0 THEN SYNC:GOTO 90      :REM If error, resync and try again
```

TASK Defines an independent task at a specified line number. Tasks are executed concurrently with a normal BASIC program (multitasking).

Syntax : TASK line number

Line number defines the first line-number of a separate BASIC program, that will be executed concurrently with the normal program.

A maximum of three (3) additional tasks may be specified. Note that the first task is implied to be the first line of the program.

Example

```
10    TASK 1000                  :REM Task 2 starts here
20    TASK 2000                  :REM Task 3 starts here
30    TASK 3000                  :REM Task 4 starts here
40    REM ____________________
50    REM Start of Task 1
60    REM ____________________
70    SUSPEND 100:GOTO 70        :REM loop here

1000  REM ----------------------------------------
1010  REM Start of Task 2
1020  REM ____________________
1030  SUSPEND 100:GOTO 1030      :REM loop here

2000  REM ____________________
2010  REM Start of Task 3
2020  REM ____________---_____
2030  SUSPEND 100:GOTO 2030      :REM loop here

3000  REM ____________________
3010  REM Start of Task 4
```

In this way, for each port an independently running program can be defined.

The PAUSE or INPUT statement will only halt execution of the task, in which they are declared, all other tasks will continue. This allows, for example, a terminal that displays the actual proces status, and a terminal that performs an interactive task with an operator.

The multitasking operating system can interrupt a TASK at any time. For that reason, the programmer must be aware that certain routines may be interrupted in the middle of an algorithm. POKE and PEEK statements (and other statements that directly manipulate memory) may cause interaction with other running TASKs. However, the currently active PORT statement will be saved and restored during task switching.

For a system example, see appendix E (page 81).

TIME Prints current time out the selected serial port

Syntax : TIME

For more information on the real time clock, refer to Appendix H (page 89).

UPLOAD Loads a program from a standard PROM programmer into memory. The PROM programmer should be connected to port B.

Syntax : UPLOAD

The opposite command for UPLOAD is DWLOAD.

The UPLOAD command should be given, before the PROM programmer is placed in the UPLOAD mode.

See also DWLOAD (page 39) and Appendix I (page 91).

VAL Returns the numerical value of a string expression.

Syntax : var = VAL (string expression)

If the first characters of the string expression are not numeric, VAL (A$) will return zero.

EXAMPLES

```
A = VAL ("12"+"34")                    (A gets the value 1234)
PRINT VAL("$AF")                       (Prints 175)
PRINT VAL ("100 PARK AVENUE")          (Prints 100)
A$="AB12CD":PRINT VAL (MID$(A$,3,2))   (Prints 12)
```