Chapter 7 BASIC Statements

+	Adds two numbers
-	Subtract two numbers
*	Multiply two numbers
/	Divide two numbers
.	AND two integers (bytes)
^	OR two integers (bytes) (ASCII $5E)
I	XOR two integers (bytes)
AUTO ON/ AUTO OFF	Sets or resets the auto re-start function (convenient in cases of a power-failure).
BREAD	Reads a block of Holding registers from a PLC.
BWRIT	Writes a block of Holding registers to a PLC.
CLOSE	Redirects the input from port A.
CRC	Calculates a CRC-16 over an array.
DATE	Prints today's date out the selected serial port
DIM	Allocates memory for arrays (subscripted variable)
END	Terminates program execution and returns to the prompt mode.
EXEC	Transfers control to a machine language program at a specified address.
FOR..NEXT	Defines a program loop.
GET	Reads characters from the input buffer into a variable.
GOSUB	Subroutine jump.
GOT0	Program jump to line number.
HR(x)	Read or write Holding Register in PLC
IF..THEN	Conditional test and action upon.
IG(x)	Read or write Input Group from I/O bus
INPUT	Reads characters from the input buffer.
INSTR	Tests the occurrence of the second string in the first string.
IR(x)	Read or write Input Register from I/O bus

LED2OFF	Turns the module status LED 2 (lower) off.
LED2ON	Turns the module status LED 2 (lower) on briefly.
LEDOFF	Turns the module status LED 1 (upper) off.
LEDON	Turns the module status LED 1 (upper) on.
LET	Assignment statement.
LINE INPUT	Input statement for string variables, ignores comma's
LOC	Returns the number of characters in the associated input buffer.
MBRCV	Modbus function 3 (register read) command.
MBXMT	Modbus function 16 (register write) command.
MID$	Copies a part of a string, into another string variable.
NEXT	Indicates end of a program loop.
OG(x)	Read Output Group from I/O bus
OPEN	Opens port B for data input from tape.
OR(x)	Read Output Register from I/O bus
PAUSE	Holds program execution until a carriage return is entered.
PEEK	Reads the value of a specified address.
POKE	Writes a value into a specified address.
PORT	Redirects input/output to another port, sets baud and data format.
PRHEX	Prints data of numerical values in hexadecimal format.
PRINT	General output statement, numerical values are printed in decimal format.
REM	Allows for remarks. Control is given to the next program line.
RND	Generates a pseudo-random number.
RETURN	Returns from subroutine.
SETDATE	Change real time clock DDD MM/DD/YR
SETTIME	Change real time clock HH:MM:SS
SUSPEND	Suspends execution of a task for a specified number of system clock ticks (20 mSec.).

SYNC	Synchronize with a Westinghouse PLC that uses 6 byte protocol.
TAB	Used in a print statement, positions the cursor at the location specified by an expression.
TASK	Defines a task at a line number. The automatic task scheduling provides concurrent execution of several (max.4) tasks.
TIME	Prints current time out the selected serial port.
VAL	Converts a string into a number.

Chapter 8 Alphabetic Function Description

This chapter describes the syntax of each statement or command. Examples are given to illustrate the use of a statement.

AUTO Controls the automatic restart of the program after a power failure.

Syntax : AUTO ON sets automatic restart.
AUTO OFF resets automatic restart.

Once the AUTO ON statement is executed the module is forced into 'run' after a power failure, Unlike the RUN command however, variables declared by **a** DIM statement are preserved. All other variables are cleared. This function is not necessary if the user program is stored in EPROM. Programs are automatically reloaded from EPROM and started on initial power-up.

EXAMPLE.

```
 5    AUTO ON
10    DIM A(10)
15    IF A(10) = 0 GOT0 25
20    PRINT 'START-UP AFTER POWER FAILURE"
25    A(10) = 5
30    PRINT "PRESS RESET BUTTON"
40    FOR B = 1 TO 1000 : NEXT B : GOT0 40
```

The RESET button (on the front of the **module)** performs different tasks depending on the use of the AUTO ON statement:

Program burned into PROM

AUTO ON present Module reloads **program** and restarts, regardless of whether the program was previously running or was stopped.

AUTO ON **not present** Module reloads program and restarts, regardless of whether the program was previously running or was stopped.

Program NOT burned into PROM (and program already running)

AUTO ON present Module restarts the program from the beginning.

AUTO ON not present Module terminates program and returns to the COMMAND (Immediate) mode.

Program NOT burned into PROM (and program stopped in COMMAND mode)

AUTO ON present Module stays in COMMAND mode.

AUTO ON not present Module stays in COMMAND mode.

BREAD Reads a block of Holding Registers from a PC700, PC900, PC1 100, PC1200 or PC1250 using the 6 byte protocol (Opcode 5).

Syntax : BREAD (start, number, array, error)

start	first register number to read (1 - 32767 or 1 to $FFFF)
number	number of registers to read (1 - 64)
array	location to place data (array variable)
error	error code (0 - no errors)

EXAMPLE

```
100   DIM H(64)
110   A=1:B=64                      :REM Preset registers
120   BREADA,B,H(1),E               :REM Read HRl-HR64, put data in H(l)-H(64)
130   IF Ec >0 THEN GOT0 120        :REM If error occurred, retry
```

EXAMPLE

```
100   DIM A(64)
110   BREAD 1,30, A(l), E           :REM HRl-HR30, data in A(l)-A(30)
```

The BREAD function requires that the serial port Request to Send (RTS) lead be raised high. Depending on the version of BASIC you are using, this may be the default or it may require that your program manually raise it. See Special Locations in EPROM page 95. To raise the RTS lead through program control refer to Appendix B (page 74). If your program 'hangs' (stops) on this statement, most likely the RTS lead for that port is low.

See also BWRIT, SYNC, MBRCV, MBXMT, EXEC $BE00 (claim), EXEC $BF00 (release). This function interacts with certain option packages (Modbus, Port Share).

For more information on the Westinghouse Program Loader protocol (6 byte), refer to the Westinghouse Programmable Controller Communications Manual (NLAM-B58).

BWRIT Writes a block of Holding Registers to a PC700, PC900, PC1 100, PC1200 or PC1250 using the 6 byte protocol (Opcode 7).

Syntax : BWRIT (start, number, array, error)

start		first register number to write (1 - 32767)
number	-	number of registers to write (1 - 64)
array		data to transmit (array variable)
error		error code (0 - no errors)

EXAMPLE

```
100  DIM H(64)
105  H(1)=44                      :REM Preset registers
106  H(2)=55
110  A=1:B=2
120  BWRIT A,B,H(I),E             :REM Write H(1) and H(2) into HR1 and HR2
130  IF E<>0 THEN GOT0 120        :REM If error occurred, retry
```

EXAMPLE

```
100  DIM A(64)
110  BWRIT 1,30, A(I), E   :REM write A(1) through A(30) into
                           :REM HR0001 to HR0030
```

See also BREAD, SYNC, MBRCV, MBXMT, EXEC $BEOO (claim), EXEC $BFOO (release). This function interacts with certain option packages (Modbus, Port Share).

The BREAD function requires that the serial port Request to Send (RTS) lead is raised high. Depending on the version of BASIC you are using, this may be the default or it may require that your program manually raise it. See Special Locations in EPROM page 95. To raise the RTS lead through program control refer to Appendix B (page 74). If your program 'hangs' (stops) on this statement, most likely the RTS lead for that port is low.

For more information on the Westinghouse Program Loader protocol (6 byte), refer to the Westinghouse Programmable Controller Communications Manual (NLAM-B58).

CLOSE Terminates data load from tape and redirects input from port A.

Syntax : CLOSE

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

CRC Calculates the CRC-16 16 bit checksum

Syntax : CRC count,databuffer,crc

count - number of bytes in array (max 255)
databuffer - location of data to calculate CRC over
crc - variable that contains CRC after execution

EXAMPLE

```
100  DIM A(255),B(2)
120  A(1)=1 : A(2)=33 : A(3) =43
130  CRC 3,A(1),B(1)                :REM Find CRC, put into B(1) and B(2)
```

Purpose of CRC Algorithm

When data is sent from one computer to another the question arises, was the information received correctly? One method of verifying that data has not been corrupted is to send the data twice. The receiving computer can compare both messages. If they are the same, then it is likely that the data has been received without error.

Sending data twice is, however, wasteful of channel bandwidth. Another way of verifying a message is via the checksum method. In the checksum method, a number is calculated that is equal to the sum of all the bytes in the message. This additional number is sent at the end of the message. The receiving computer recalculates the checksum and compares this value with the last byte of the message.

Although the checksum uses less bandwidth, it is effective. A more sophisticated checksum is called the Cyclic Redundancy Check (CRC). The CRC-1 6 standard is used by several communications protocols (including Modbus RTU).

The CRC function could, therefore, be used to create additional Modbus function code messages (in addition to function 3 and 16 supported by MBXMT and MBRCV.

DATE Prints today's date out the selected serial port.

Syntax : DATE

Use the PORT command to select the serial port.

DIM Specifies the maximum values for array variable subscripts and allocates storage memory accordingly.

Syntax : DIM var(subscript) [,var(subscript)]

Var is the name of the array (A...Z).

Subscript is one expression or two expressions separated by a comma, which define the dimensions of the array.

The interpreter does not recognize subscripted variables, that have not been declared by a DIM statement before.

The minimum value for a subscript is 1, the maximum value is 255.

Variables that have been DIMentioned, are not initialized to zero after a power failure (assuming the AUTO ON statement has been used, and the program has not been stored in EPROM)

EXAMPLE

```
10    PRINT 'ARRAY STORAGE EXAMPLE'
20    DIM A(200,2)
30    FOR B = 1 TO 20
40    FOR C = 1 TO 2
50    A(B,C) = B*C
60    NEXTC:NEXTB
70    PRINT "ARRAY FILLED' : SIZE
```

DWLOAD Down-loads the BASIC program in memory to a PROM programmer, with the use of Motorola's Exorciser format (block size $10000, offset 0, start address 0), to port B.

Syntax : DWLOAD

The PROM programmer must be in the down-load mode, before the DWLOAD command is executed.

The purpose of this command is to produce an EPROM back-up of the program, provided the size of the program is less than 16k bytes.

Upon start-up, the program in EPROM is automatically transferred to RAM memory and will start automatically. The use of the AUTO statement is in this case superfluous.

The EPROM program is also transferred whenever:

1. Exiting the EDITOR
2. RESET pushbutton pushed

It is also possible to have more than 1 program stored in the EPROM. The user can select these programs by using the PROGRAM statement.

All programs must be downloaded to consecutive memory locations. The first four bytes contain the programs start and end address specification,

Refer to appendix I (page 91) for more information.

EDIT Invokes the line editor, see appendix A (page 67) for detailed information,

Syntax : EDIT

END Terminates program execution, and returns to the prompt mode (port A).

Syntax : END

The END statement may be placed anywhere in the program.

EXAMPLE

```
10    IF K>1000 THEN END
20    K=K+1 : PRINT 'A';: GOT0 10
```

Above program prints one thousand A's, then returns to prompt mode.

There is no way to automatically restart a program from the prompt mode. If a program was originally burned into EPROM, cycling power will reload RAM and restart the program.

The prompt message 'READY #' will be sent out the last selected serial port. The prompt will be sent at the baud and data format for that particular port.

EXEC Transfers control to a subroutine in machine language. The starting address of the subroutine must be specified by an expression.

Syntax : EXEC expression.

For use of the EXEC function, refer to the memory map and detailed information in appendix B.

The module can execute an 6809 assembly language program that was loaded into RAM (using POKE statements) or else directly out of EPROM.

This statement forces an absolute microprocessor jump to the address specified. If the jump address is invalid or the code found at that point is corrupted, a watchdog timer trip will occur.

Example EXEC locations

EXEC $BEOO claim serial port
EXEC $BF00 release serial port

For more information, refer to any 6809 reference book or 6809 compiler manual, or contact Westinghouse

FOR..NEXT Performs a series of instructions in a loop a given number of times,

Syntax : FOR var = X TO Y
NEXT var.

var is an integer or single-byte variable -like A or A$(1) - to be used as the counter.

X is a numeric expression to initialize the counter var.

Y is a numeric expression which is the final value of the counter var.

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

The program lines between FOR and NEXT are executed repeatedly, until var is greater than Y. Each loop, var is incremented by one by the NEXT statement.

EXAMPLE

```
5   B = 6
10  FOR C = 1 TO B/3 : PRINT C : NEXT C
```

GET Reads a specified number of characters from the input buffer.

Syntax : GET var

or GET stringvar, N.

GET var will read 1 character from the input buffer, and var will be assigned to the decimal value of that character.

GET stringvar, N will read N characters from the input buffer, and copy them into the string variable. The maximum length is 80.

The number of characters available in the input buffer can be tested by the LOC statement.

Warning: Executing the statement:

GET N$,LOC (N$ is any string variable)

will result in an error if LOC > 80. Since a receive buffer holds up to 127 bytes, this may happen if data is not removed from the buffer as fast as it is placed there.

As a character is received on each of the four serial ports, a background task automatically places this character into a 127 byte circular buffer. The GET statement removes from one to 80'bytes from the buffer at a time, and places this data into a BASIC variable. Each serial port has its own circular buffer.

EXAMPLE

```
10  PRINT 'GET EXAMPLE'
20  PRINT 'STRIKE ANY KEY TO CONTINUE"
30  GET A : PRINT #A,"DECIMAL VALUE IS ';A
40  PRINT "ENTER 6 CHARACTERS'
50  IF LOC>5 GOT0 90
60  PRINT 'WAITING' :FOR C=I TO 2500:NEXT C:GOTO 50
90  GET A$,6 : PRINT A$
```

GOSUB Branches to a subroutine.

Syntax : GOSUB expr.

Expr. is numerical expression that results in a line number of a subroutine (computed GOSUB).

Subroutines can be nested up to a depth of 8.

Control is given to the line number, calculated by an expression. A RETURN statement causes BASIC to branch back to the statement, following the most recent GOSUB statement.

EXAMPLE

```
10    INPUT 'ENTER ALARM NUMBER (1 OR 2)';A
20    IF A>2 GOT0 100
30    IF A<1 GOT0 100
40    GOSUB A*10+40 : GOT0 10
50    PRINT "NUMBER 1 ALARM" : RETURN
60    PRINT 'NUMBER 2 ALARM' : RETURN
100   PRINT 'WRONG NUMBER' : RUN
```