PUT

Format

PUT[#]<file number>[,<record number>]

Purpose

Writes the contents of a random file buffer to one record of a random access file.

Remarks

The random access file must be opened in the "R" mode under the number specified in <file number> before this statement can be executed. Further, data must be set in the random file buffer with the LSET and/or RSET statements.

<file number> is the number under which the file was opened, and <record number> is the number of the file record to which the buffer contents are to be written.

<record number> must have a value in the range from 1 to 32767. If <record number> is omitted, the contents of the random file buffer will be written to the record following the record accessed by the previous PUT or GET statement.

See also

GET, LSET/RSET, OPEN and Chapter 6.

Example

```
10 OPEN "R",#1,"RANDOM",20
20 FIELD #1,10 AS A$
30 LSET A$="ABCDEFGHIJ"
40 PUT #1
50 WRITE#1,1,2,"YZa"
60 PUT #1
70 PRINT#1,"KLMNOPQ";
80 PUT #1
90 FOR I=1 TO 3
100 GET #1,I
110 PRINT A$
120 NEXT

run
ABCDEFGHIJ
1,2,"YZa"
KLMNOPQa"
Ok
```

NOTE:
String data to be written to a random access file with PUT can be placed in the random file buffer with the PRINT#, PRINT# USING, and WRITE# statements, as well as with the LSET/RSET statements. An example of this is INCLUDED IN THE PROGRAM ABOVE. When the WRITE# statement is used for this purpose, extra positions in the buffer are padded with spaces. An FO error (Field overflow) will occur if an attempt is made to read or write past the end of the buffer.

RANDOMIZE

Format

RANDOMIZE [<expression>]

Purpose

Reinitializes the sequence of random numbers generated by the RND function using the seed number specified in <expression>.

Remarks

The value specified in <expression> must be a number in the range from −32768 to 32767. If <expression> is omitted, BASIC suspends program execution and displays the following message to prompt the operator to enter a value from the keyboard.

Random number seed ?

If the random number sequence is not reinitialized, the RND function will return the same sequence of random numbers each time a given program is executed. This can be overcome by placing a RANDOMIZE command at the beginning of each program so that the user can input a random number. It is better however, if the computer changes the random number seed in <expression>. This can be achieved by using the TIME$ function as this is a continually changing string of numbers. A description of this is given in the example program in RND.

See also

RND

Example

```
10 FOR J=1 TO 3
20 RANDOMIZE
30 PRINT
40 FOR I=1 TO 5
50 PRINT RND;
60 NEXT I
70 PRINT
80 NEXT J

run
Random number seed (-32768 to 32767)? 1
 .58041  .128928  .928324  .901162  .532818
Random number seed (-32768 to 32767)? 2
 .89341  .823736  .964563  .674916  .963391
Random number seed (-32768 to 32767)? 1
 .826124  .915422  .0593067  .381003  .511101
Ok
```

READ

Format

READ <list of variables>

Purpose

Reads values from DATA statements and assigns them to variables.

Remarks

The READ statement must always be used in conjunction with one or more DATA statements. The READ statement assigns items from the <list of constants> of DATA statements to variables specified in <list of variables>. Items from the <list of constants> are substituted into variables in the <list of variables> on a one-to-one basis, and the type of each variable to which data is assigned must be the same as the type of the corresponding constant in <list of constants>.

A single READ statement may access one or more DATA statements, or several READ statements may access the same DATA statement.

If the number of variables specified in <list of variables> is greater than the number of constants specified by DATA statements, an OD error (Out of data) will occur. If the number of variables specified in <list of variables> is smaller than the number of constants specified in DATA statements, subsequent READ statements begin reading data at the first item which has not previously been read. If there are no subsequent READ statements, the extra items are ignored.

The next item to be read by a READ statement can be reset to the beginning of the first DATA statement on any specified line by means of the RESTORE statement.

See also

DATA, RESTORE

Example

```
10 FOR J=1 TO 5
20 READ A(J),B$(J),C(J)
30 NEXT J
40 FOR J=1 TO 5
50 PRINT A(J),B$(J),C(J)
60 NEXT J
70 END
80 DATA 1,aaaa,11
90 DATA 2,bbbb,22
100 DATA 3,cccc,33
110 DATA 4,dddd,44
120 DATA 5,eeee,55

run
1           aaaa           11
2           bbbb           22
3           cccc           33
4           dddd           44
5           eeee           55
Ok
```

REM

Format

REM <remark>
' <remark>

Purpose

Makes it possible to insert explanatory remarks into programs.

Remarks

Either of the above formats may be used to insert explanatory remarks into a program. Remark statements are ignored by BASIC during program execution, but are output exactly as entered when the program is listed.

If program execution is branched to a line which begins with a remark statement, execution resumes with the first subsequent line which contains an executable statement.

If a remark statement is to be appended to a line which includes an executable statement, be sure to precede it with a colon (:). Also, note that any executable statements following a remark statement on a given program line will be ignored.

NOTE:
When a program is listed using LIST* or LLIST*, apostrophes indicating remark statements are not output. However, "REM" is output at the beginning of any remark statements beginning with REM.

REMOVE

REMOVE

Writes the directory to the microcassette tape and terminates microcassette I/O.

A REMOVE command must be executed before taking a cassette tape out of the microcassette drive. The reason for this is that, if the cassette tape is taken out of the drive without executing the REMOVE command, the directory is not updated and data which has been written to the cassette up to that point may be lost. Further, if another tape is inserted in the microcassette drive without executing the REMOVE command for the previous tape, the contents of the new tape may be destroyed.

The REMOVE command cannot be executed if any files are open or the tape in the drive has not been mounted.

DW error (Disk write error) — An error occurred while data was being written to the microcassette drive.

AC error (Tape access error) — The REMOVE command was executed before the tape in the drive had been installed by executing the MOUNT command; or, the REMOVE command was executed while a tape file was still open.

MOUNT
Section 2.4.5

RENUM

**RENUM[[<new line number>][,[<old line number>]
[,<increment>]]]**

Renumbers the lines of programs.

This command renumbers the lines of a program according to the values specified in <new line number>, <old line number>, and <increment>. <new line number> is the first line number to be used in the new sequence of program lines and <old line number> is the line number in the current program with which renumbering is to begin. <increment> is the amount by which each successive line number in the new sequence is to be increased over the number of the preceding line.

The default value for <new line number> is 10, that for <old line number> is the first line of the current program, and that for <increment> is 10.

The RENUM command also changes all line number references included in GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB and ERL statements to reflect the new line numbers. If a nonexistent line number appears after one of these statements, the message "UL error in xxxxx" is displayed. The incorrect line number in the program line is not changed, but that indicated by xxxxx may be changed.

RENUM
Renumbers the entire program. The first line number of the new sequence will be 10, and the numbers of subsequent lines will be increased in increments of 10.

RENUM 300,50
Renumbers program lines starting with existing line 50. The number of line number 50 in the current program will be changed to 300, and all subsequent lines will be increased in increments of 10.

RENUM 1000,900,20
Renumbers the lines beginning with 900 so that they start with line number 1000 and are increased in increments of 20.

NOTE:
The RENUM command cannot be used to change the order of program lines
(for example, RENUM 15, 30 when the program has three lines numbered 10,
20 and 30), or to create line numbers greater than 65529. An FC error (Illegal
function call) will result if this rule is not observed.

RESET

Format   **RESET**

Purpose   Resets the READ/ONLY condition which results when the floppy disk in a disk drive has been exchanged for another one. When the floppy disk in an external disk drive is replaced with another one when that drive has previously been accessed, subsequent writes to that drive are inhibited. This is to protect the contents of the disk's directory. Executing the RESET command resets the read-only condition and re-enables access to the new disk. It also closes any files which are open in the same manner as the CLOSE command.

Also RESET enables a new ROM capsule for read access after replacement.

The default drive is the default drive for CP/M until a RESET command is executed. The execution of a RESET command sets the default drive to drive A: and so programs should specify the drive to which the data is to be saved.

Example
```
100 CLOSE
110 PRINT "Replace disk in drive E: and press
    enter when ready"
120 A$=INPUT$(1)
130 RESET
140 OPEN"O",#1,"E:FILE1"
```

RESTORE

RESTORE [<line number>]

Allows DATA statements to be re-read from a specified program line.

If <line number> is specified, the next READ statement will access the first item in the DATA statement on the specified line or on the first subsequent line which contains a DATA statement if there is no DATA statement in the specified line. If <line number> is not specified, the next READ statement accesses the first item in the first DATA statement in the program.

DATA, READ

```
10 FOR I=1 TO 2
20 FOR X=1 TO 8
30 READ A:SOUND A,50
40 NEXT X,I
50 SOUND 0,100:RESTORE
60 FOR I=1 TO 2
70 FOR X=1 TO 8
80 READ A:SOUND A*2,50
90 NEXT X,I
100 DATA 256,288,320,341,384,426,480,512
110 DATA 512,480,426,384,341,320,288,256
```

RESUME

RESUME
RESUME 0
RESUME NEXT
RESUME <line number>

Used to continue program execution after execution has branched to an error processing routine.

The RESUME statement makes it possible to resume program execution at a specific line or statement after error recovery processing has been completed. The point at which execution is resumed is determined by the format in which the statement is executed as follows:

RESUME   Resumes program execution at the statement
or   which caused the error.
RESUME 0

RESUME NEXT Resumes program execution at the statement immediately following that which caused the error.

RESUME   Resumes program execution at the program line
<line number> specified in <line number>.

An RW error (RESUME without error) message will be generated if a RESUME statement is encountered anywhere in a program except in an error processing routine.

ERROR, ON ERROR GOTO, ERR/ERL

See the example program under ERROR.

RIGHT$

Format   RIGHT$(X$,J)

Purpose   Returns a string composed of the J characters making up the right
hand end of string X$.

Remarks   The value specified for J must be in the range from 0 to 255.
If J is greater than the length of X$ the entire string will be
returned. If J is equal to 0 a null string of zero length is returned.

See also LEFT$, MID$

Example
```
10 A$="Epson PX-4"
20 FOR I=1 TO 10
30 PRINT RIGHT$(A$,I)
40 NEXT
run
4
-4
X-4
PX-4
 PX-4
n PX-4
on PX-4
son PX-4
pson PX-4
Epson PX-4
Ok
```

RND

Format   RND[(X)]

Purpose   Returns a random number with a value between 0 and 1.

Remarks   RND returns a random number from a sequence determined
mathematically by the random number generator in the BASIC
interpreter. The same sequence of numbers is generated each time
a program containing the RND function is executed. If X is omit-
ted or the number specified for X is greater than 0, the next ran-
dom number in the sequence is generated. If 0 is specified for X,
RND repeats the last random number generated. If the number
specified for X is less than 0, RND starts a new sequence whose
initial value is determined by the value specified for X.
It is sometimes necessary to generate numbers in a given range.
The following examples show how this may be done.

INT (RND (X) * 100)	Generates numbers in the range 0—99.
INT (RND (X) * 100) + 1	Generates numbers in the range 1—100.
INT (RND (X) * 100) + 50	Generates numbers in the range 100—149.
INT (RND (X) * 10) - 5	Generates numbers in the range -5 to +4.

See also RANDOMIZE

Example   The following program shows how to use RANDOMIZE and the
value returned by TIME$, to give a random number which is as
random as the computer will allow. Run the program a number
of times to see that the numbers output from line 30 are the same
each time the program is run. The first value will be the same
for the five circuits of the repeating loop (lines 30 to 50) because
a negative value of X repeats the number by continually reseed-
ing with the same number. The next values will be the same as
these values because X = 0 which repeats the last random num-
ber. When X is from 1 to 3 a different number is produced each
time in the loop lines 30 - 50, but this number will be the same
each time the program is run.

The second set of numbers, generated by lines 100 – 130, is again the same every time the program is run, despite RANDOMIZE having been used.

The last set of numbers are different every time the program is run because the number used to generate the seed is different. It is obtained from the string returned by TIME$, by multiplying the hours by the seconds. This requires not only string manipulation to remove the characters from each end of the string, but also using the VAL function to convert them into numbers. More complex algorithms could be used.

```
10 FOR X = -1 TO 3
20 FOR N = 1 TO 5
30 PRINT RND(X);
40 NEXT N
45 PRINT
50 NEXT X
60 '
70 '
80 ' first randomize routine
90 RANDOMIZE(456)
100 FOR J = 1 TO 10
110 PRINT INT(RND(1)*1000);
120 NEXT
130 PRINT
140 '
150 ' second randomize routine
160 FOR J = 1 TO 10
170 RANDOMIZE(VAL(LEFT$(TIME$,2))*VAL(RIGHT$(TIME$,2)))
180 PRINT INT(RND(1)*1000);
190 NEXT
200 PRINT
```

```
 .308601   .308601   .308601   .308601   .308601
 .308601   .308601   .308601   .308601   .308601
 .498871   .670127   .98706    .739354   .783018
 .949844   .55241    .681371   .823571   .244878
 .421504   .775332   .310637   .346463   .056878
 422   292   906   614   667   833   595   910   875   173
 86   120   948   4   839   687   0   507   224   514
Ok
```

RUN

Format

RUN [<line number>]
RUN <file descriptor>[,R]

Purpose

Initiates program execution.

Remarks

The first format is used to start execution of the program in the currently selected program area. Execution begins at the first line of the program unless <line number> is specified. If <line number> is specified, execution begins at the specified line. All files are closed and variables cleared, even if the <line number> specified is not the first line of the program. To restart a program from a particular line number without using RUN, use GOTO <line number>.

The second format is used to load and execute a program from a disk device, including the microcassette drive and RS-232C interface. Specify the name under which the program was saved in <file descriptor>; if the extension is omitted, ".BAS" is assumed. For the RS-232C interface, specify "COMØ:[(<options>)]" as the file descriptor.

The RUN command normally closes all files which are open and deletes the current contents of memory before loading the specified program. However, all data files will remain open if the R option is specified although the variables will be cleared.

See also

GOTO, LOAD, MERGE

Examples

RUN 300
runs the program from line 300.

RUN "ADDRESS.BAS"
loads and runs the program "ADDRESS.BAS" in the default drive.

RUN "D:ENTRY.BAS",R
loads and runs the program "ENTRY.BAS" from disk drive D: without closing the files which were opened by the previous program.

SAVE

Format

SAVE <file descriptor>[,A]
SAVE <file descriptor>[,P]

Purpose

Used to save programs to disk device files or the RS-232C communications interface.

Remarks

This command saves BASIC programs to disk files or the RS-232C communications interface. In the former case, specify the drive name, file name and extension in <file descriptor>.
The currently active drive is assumed if the drive name is omitted, and ".BAS" is assumed if the extension is omitted. In the latter case, specify "COMØ:[(<options>)]" as the file descriptor.

If the A option is specified, the program will be saved in ASCII format; otherwise it will be saved in compressed binary format. The ASCII format requires more disk space for storage than binary format, but some file access operations require that the file be in ASCII format (for example, the file must be in ASCII format if it is to be loaded with the MERGE command).

If the P (protect) option is specified, the program will be saved in an encoded binary format. When a file is saved using this option it cannot be edited or listed when it is subsequently loaded. Once a program has been saved with the P option the protected condition cannot be cancelled.

See also

LOAD, MERGE

Examples

SAVE "ADDRESS"
SAVE "B:ADDRESS.ASC",A
SAVE "COMØ:(A8N3FXN)"
SAVE "SECRET",P

SCREEN

Format

SCREEN (<horizontal position>,<vertical position>)

Purpose

Returns the character code of the character displayed at the specified position on the screen.

Remarks

The SCREEN function returns the value of character code for the character displayed at the virtual screen coordinates specified by <horizontal position> and <vertical position>. <horizontal position> must be specified as a numerical expression with a value in the range from 1 to Xmax; <vertical position> is also specified as a numerical expression, but with a value in the range from 1 to Ymax. The values of Xmax and Ymax are determined by the size of the screen currently being used as the write screen.

FC error (Illegal function call) — An argument specified was not in the prescribed range.

SGN

Format

SGN(X)

Purpose

Returns the sign of numeric expression X.

Remarks

If X is greater than 0, SGN(X) returns 1. If X equals 0, SGN(X)
returns 0. If X is less than 0, SGN(X) returns −1. Any numeric
expression can be specified for X.

Example

```
10 A=1:  PRINT SGN(A)
20 B=1<0:PRINT SGN(B)
30 C=1=1:PRINT SGN(C)

run
 1
 0
-1
Ok
```

SIN

Format

SIN(X)

Purpose

Returns the sine of X, where X is an angle in radians.

Remarks

The sine of angle X is calculated to the precision of the type of
the numeric expression specified for X.

Example

```
10 CLS
20 INPUT "Enter angle in degrees";A
30 PI=4*ATN(1)
40 D=PI/180
50 PRINT "SIN(";A;")=";SIN(A*D)
60 GOTO 20

Enter angle in degrees? 0
SIN( 0 )= 0
Enter angle in degrees? 30
SIN( 30 )= .5
Enter angle in degrees? 45
SIN( 45 )= .707107
Enter angle in degrees?
```

SOUND

SOUND <pitch>, <duration>

Generates a sound of the specified pitch and duration.

The <pitch> parameter is specified as a value from 0 to 4500. Values from 100 to 4500 cause a sound to be generated at the equivalent pitch, and values from 0 to 99 result in no sound.

The <duration> parameter is specified as a value from 0 to 2554; the length of the sound generated is equal to <duration> × 10 msec, with the result rounded off to the nearest millisecond.

When a SOUND statement is executed, the BASIC interpreter waits for execution of that statement to be completed before going on to the next statement.

The relationship between values specified in <pitch> and the notes of the musical scale is as shown in the table below.

Unit: Hz

Note	1	2	3	4	5
C	130	261	523	1046	2093
C#	138	277	554	1108	2217
D	146	293	587	1174	2349
D#	155	311	622	1214	2489
E	164	329	659	1318	2637
F	174	349	698	1396	2793
F#	184	369	739	1479	2959
G	195	391	783	1567	3135
G#	207	415	830	1661	3322
A	220	440	880	1760	3520
A#	233	466	932	1864	3729
B	246	493	987	1975	3951

FC error (Illegal function call) — A parameter specified was outside the permissable range.

MO error (Missing operand) — A required operand was not specified in the statement.

SPACE$

SPACE$(J)

Returns a string of spaces whose length is determined by the value specified for J.

The value of J must be in the range from 0 to 255. If other than an integer expression is specified for J, it is rounded to the nearest integer.

SPC

```
10 FOR J=1 TO 7
20 READ A$,B$
30 P$=A$+SPACE$(20-LEN(A$+B$))+B$
40 PRINT P$
50 NEXT
60 DATA Angie,523-2121,Alfie,456-1010,Robert,21-4444
70 DATA Susan,223-1234,Charlie,234-2324,John,703-7654
80 DATA Randolph,631-1360

run
Angie        523-2121
Alfie        456-1010
Robert        21-4444
Susan        223-1234
Charlie      234-2324
John         703-7654
Randolph     631-1360
Ok
```

SPC

SPC(J)

Returns a string of spaces for output to the display or printer.

The SPC function can only be used with an output statement such as PRINT or LPRINT — unlike the SPACE$ function, it cannot be used to assign spaces to variables. The value specified for J must be in the range from 0 to 255.

SPACE$

```
10 FOR J=1 TO 7
20 READ A$,B$
30 PRINT A$;SPC(20-LEN(A$+B$));B$
40 NEXT
50 DATA Angie,523-2121,Alfie,456-1010,Robert,21-4444,Susan,2
23-1234,Charlie,234-2324,John,703-7654,Randolph,631-1360

run
Angie        523-2121
Alfie        456-1010
Robert        21-4444
Susan        223-1234
Charlie      234-2324
John         703-7654
Randolph     631-1360
```

SQR

Format　　SQR(X)

Purpose　　Returns the square root of X.

Remarks　　The value specified for X must be greater than or equal to 0.

Example
```
10 PRINT "X","SQR(X)"
20 FOR X=0 TO 100 STEP 10
30 PRINT X,SQR(X)
40 NEXT
run
X                 SQR(X)
 0                0
 10               3.16228
 20               4.47214
 30               5.47723
 40               6.32456
 50               7.07107
 60               7.74597
 70               8.3666
 80               8.94427
 90               9.48683
 100              10
Ok
```

STAT

Format　　STAT [| <program area no.> |]
　　　　　　　　　　| ALL |

Purpose　　Displays the status of program areas.

Remarks　　Executing the STAT command without specifying any parameters displays the status of the currently selected program area. If <program area no.> is specified, the status of the specified program area is displayed. In both cases, the display format is as follows.

Pn:XXXXXXXX　　YYYYY Bytes

Here, n indicates the number of the applicable program area, XXXXXXXX indicates the name assigned to that program area by the TITLE command, and YYYYY indicates the size of the program area (i.e., the size of the program in that area) in bytes. Spaces are displayed for XXXXXXXX if no name has been assigned with the TITLE command.

When an asterisk ("＊") is displayed between Pn and the program area name, the edit inhibit attribute has been set for that area with the "TITLE ...,P" command.

Executing STAT ALL displays the status of all program areas as follows.

P1:AAAAAAAA　　aaaaa　Bytes
P2:BBBBBBBB　　bbbbb　Bytes
P3:CCCCCCCC　　ccccc　Bytes
P4:DDDDDDDD　　ddddd　Bytes
P5:EEEEEEEE　　eeeee　Bytes
　　xxxxx Bytes free

The number displayed for xxxxx indicates the current number of bytes of unused memory.

FC error (Illegal function call) — A number other than 1 to 5 was specified in <program area no.>.

STOP

Format   STOP

Purpose   Terminates program execution and returns BASIC to the command level.

Remarks   STOP statements are generally used to interrupt program execution during debugging to allow the contents of variables to be examined or changed in the direct mode. Program execution can then be resumed by executing a CONT command.

The following message is displayed upon execution of a STOP statement:

Break in nnnnn

Unlike the END statement, no files are closed when a STOP statement is executed.

See also   CONT

Example
```
10 PRINT "Program line 10"
20 STOP
30 PRINT "Program line 20"

run
Program line 10
Break in 20
Ok
cont
Program line 20
Ok
```

STOP KEY

Format   STOP KEY | ON |
                        | OFF |

Purpose   Disables or reenables the [STOP] key.

Remarks   Executing STOP KEY OFF disables the [STOP] key and [CTRL] + [C] . This prevents processing from being interrupted if the [STOP] key is accidentally pressed during execution of an application program.

Executing STOP KEY ON reenables the [STOP] key (and [CTRL] + [C]) function after it has been disabled by executing STOP KEY OFF.

Since the [STOP] key and [CTRL] / [C] are completely disabled when STOP KEY OFF is executed, be careful to avoid executing it during program debugging.

The [STOP] key is not disabled unless a STOP KEY OFF command is executed. Once executed, the STOP KEY OFF command is cancelled and the [STOP] key reenabled by a hot start or execution of a RUN, MENU, or STOP KEY ON command.

STR$

STR$(X)

Converts numeric data to string data.

This function returns a string of ASCII characters which represent the decimal number corresponding to the value of X. X must be a numeric expression.

This function is complementary to the VAL function.

VAL

```
10 FOR X=1 TO 10
20 PRINT X;
30 NEXT
40 PRINT
50 FOR X=1 TO 10
60 A$=STR$(X)
70 PRINT A$;
80 NEXT

run
 1  2  3  4  5  6  7  8  9  10
 1 2 3 4 5 6 7 8 9 10
Ok
```

STRING$

STRING$(J, K)
STRING$(J, X$)

Returns a string of characters.

The length of the character string returned by this function is determined by the value of J. If K is specified the function returns a string of J characters whose ASCII code corresponds to the value of K. If a non-integer value is specified for K its value is rounded to the nearest integer before the string of characters is returned.

If X$ is specified this function returns a string of J characters made up of the first character of the specified string.

```
10 A$=STRING$(5,"A")
20 B$=STRING$(5,66)
30 PRINT A$:PRINT B$

run
AAAAA
BBBBB
Ok
```

SWAP

SWAP <variable 1>,<variable 2>

The SWAP statement exchanges the values of variables specified in <variable 1> and <variable 2>.

The SWAP statement may be used to exchange the values of any type of variable, but the same variable types must be specified in both <variable 1> and <variable 2>; otherwise a TM error (Type mismatch) will occur.

```
10  Using SWAP for alphabetization
20 FOR J=1 TO 5
30 READ A$(J)
40 NEXT J
50 FOR J=2 TO 5
60 IF A$(J-1)>A$(J) THEN SWAP A$(J-1),A$(J):J=1
70 NEXT J
80 FOR J=1 TO 5
90 PRINT A$(J)
100 NEXT J
110 DATA Mary,Charlie,Angie,Jane,Andy

run
Andy
Angie
Charlie
Jane
Mary
Ok
```

SYSTEM

SYSTEM

The SYSTEM command returns control from BASIC to CP/M.

The SYSTEM command ends operation of the BASIC interpreter and returns control to CP/M. All programs in the BASIC program areas are lost when this command is executed.

TAB

Format   TAB(J)

Purpose   Spaces to column J on the LCD screen or printer. If the cursor/print head is already past column J, it is spaced to that column on the next line.

Remarks   The character position on the far left side of the LCD screen or printer is column 0, and that on the far right side is the device width minus one. For the LCD screen, the device width is the number of columns determined for the currently selected screen by the SCREEN or WIDTH statement. For a printer, it is the number of columns determined by the WIDTH LPRINT statement.

If the value specified for J is greater than the device width minus one, the number of spaces generated is equal to J MOD n, where n is the device width.

In the expression

PRINT TAB (J) ; A$

the string A$ will be printed with the first character starting at position J. However, if the length of string A$ added to the value of J is greater than 81, the string will be printed on the next line.

The TAB function can only be used with the PRINT and LPRINT statements, and cannot be used to generate strings of spaces for other purposes.

Example
```
10 SCREEN 0
20 PRINT 1;2;3;4;5
30 PRINT 1;TAB(4);2;TAB(9);3;TAB(15);4;TAB(22);5

run
 1  2  3  4  5
 1  2     3     4     5
Ok
```

NOTE:
If a space is included between TAB and the opening bracket in TAB (J), PX-4 BASIC will interpret this as item J of an array with the name TAB. Rather than print the next string at position J, the value 0 will be printed because the value of all the items in this array will be 0. If J is greater than 10, a BS error (Subscript out of range) will be generated because the TAB array has not been dimensioned.

TAN

Format   TAN(X)

Purpose   Returns the tangent of X, where X is an angle in radians.

Remarks   The tangent of angle X is calculated to the precision of the type of numeric expression specified for X.

To convert an angle from degrees to radians, multiply it by 1.57080 (for single precision) or by 1.570796326794897/90 (for double precision).

See also ATN, COS, SIN

Example

```
10 INPUT "Enter angle in degrees";A
20 PRINT "Tangent";A;"degrees is";TAN(A*3.14159/180)
30 GOTO 10

Enter angle in degrees? 30
Tangent 30 degrees is .57735
Enter angle in degrees? 45
Tangent 45 degrees is .999999
Enter angle in degrees? 60
Tangent 60 degrees is 1.73205
Enter angle in degrees?
```

TAPCNT

Format   TAPCNT

Purpose   Reads or sets the value of the microcassette drive counter.

Remarks   The TAPCNT function reads the value of the microcassette drive counter. The value returned will be in the range from -32768 to 32767.

The TAPCNT function can be used at any time it is necessary to determine the counter value.

The TAPCNT function can also be used to set the counter value. However, in this case, the tape in the microcassette drive must be in the unmounted condition.

AC error (Tape access error) — An attempt was made to set the counter value while the tape in the microcassette drive was in the mounted condition.

IO error (Device I/O error) — An attempt was made to read or set the counter value when microcassette drive is not installed.

See also WIND

TIME$

TIME$

Reads the time of the built-in clock.

The TIME$ function returns the time of the built-in clock as an 8-byte character string. The format of this string is "HH:MM:SS", where HH indicates the hour (00 to 23), MM indicates the minute (00 to 59), and SS indicates the second (00 to 59).

TIME$ is a system variable and can be set by executing TIME$ = "HH:MM:SS".

TITLE

TITLE (<program area name>][,P]

Sets the name and protect attribute of the currently selected program area.

The TITLE command assigns a name to the currently selected program area. This name is specified in <program area name> as a string of from 0 to 8 letters. If more than 8 letters are specified, the ninth and following letters are ignored. After execution of this command, the specified program area name is displayed upon execution of the STAT command and in the BASIC start-up menu. Further, when a program file is loaded by executing the LOAD or RUN <filename> commands, the file name of the program loaded is set as the program area name.

The program area name can be cancelled by executing the TITLE command with a null string ("") specified for <program area name>. The program area name is also cancelled by executing the NEW command.

The program area name is not affected if the <program area name> parameter is omitted.

If the P (protect) option is specified, executing the TITLE statement sets the protect attribute for the currently selected program area. Once a program area has been protected in this manner, any attempt to edit that program or to execute a DELETE command in that program area will result in an FC error (Illegal function call).

MO error (Missing operand) — A required operand was not specified in the command.

TRON/TROFF

Format

TRON
TROFF

Purpose

Used to enable or disable the trace mode of execution.

Remarks

In the trace mode, the number of each line of a program is displayed on the screen in square brackets at the time that line is executed. This makes it possible to determine the sequence in which program lines are executed, and as such can be used with the STOP and CONT commands during program debugging.

The trace mode is enabled by executing TRON and is disabled by executing TROFF.

See also

CONT, STOP

Example

```
10 FOR I=1 TO 3:PRINT I;:NEXT
20 PRINT
30 TRON
40 FOR I=4 TO 6:PRINT I;:NEXT
50 TROFF

run
 1  2  3
[40] 4  5  6 [50]
Ok
```

USR

Format

USR [<digit>]<argument>

Purpose

Passes the value specified for <argument> to a user-written machine language routine and returns the result of that routine.

Remarks

<digit> is an integer from 0 to 9 which corresponds to the digit specified in the DEF USR statement for the machine language routine. If <digit> is omitted, USR0 is assumed.

A string or numeric expression must be specified for <argument>; this argument is passed to the machine language routine as described in Appendix G.

VAL

Format   VAL(X$)

Purpose   Converts a string composed of numeric ASCII characters into a numeric value.

Remarks   This function returns the numeric value of a character string consisting of numeric characters. The first character of string X$ must be " + ", " − ", " . ", "&" or a numeric character (a character whose ASCII code is in the range from 48 to 57); otherwise, this function returns 0.
Some examples of use of the VAL function are shown below.
(1) VAL(X$)
Returns the decimal number which corresponds to the string representation of that decimal number. X$ is composed of the characters "0" to "9" and may be preceded by " + ", " − " or " . ". Complementary to the STR$ function.
(2) VAL("&H" + X$)
Returns the decimal number which corresponds to the string representation of a hexadecimal number. X$ is composed of the characters "0" to "9" and "A" to "F". This is complementary to the HEX$ function.
(3) VAL("&O" + X$)
Returns the decimal number which corresponds to the string representation of an octal number. X$ is composed of the characters "0" to "7". This is complementary to the OCT$ function.

See also   HEX$, OCT$, STR$

Example
```
10 INPUT "Type in a hexadecimal number ";A$
20 PRINT "The decimal value of &H";A$;" using VAL(";CHR$(34)
;"&H";CHR$(34);"+X$) is ";VAL("&H"+A$)
30 INPUT "Type in an octal number ";B$
40 PRINT "The decimal value of &O";B$;" using VAL(";CHR$(34)
;"&O";CHR$(34);"+X$) is ";VAL("&O"+B$)
run
Type in a hexadecimal number ? 4F
The decimal value of &H4F using VAL("&H"+X$) is   79
Type in an octal number ? 32
The decimal value of &O32 using VAL("&O"+X$) is   26
Ok
```

VARPTR

Format   VARPTR(<variable name>)
VARPTR(# <file number>)

Purpose   The first format returns the address in memory of the first data byte of the variable specified in <variable name>.
The second format returns the starting address of the I/O buffer assigned to the file opened under <file number>.

Remarks   With the first format, a value must be assigned to <variable name> before executing VARPTR; otherwise, an "Illegal function call" error will result. Any type of variable name (numeric, string, or array) may be specified, and the address returned will be an integer in the range from − 32768 to 32767. If a negative number is returned, add it to 65536 to obtain the actual address.

Storage of the various types of data in memory is as follows. (↓ indicates the byte which corresponds to the value returned by VARPTR.)

(1) Integer variables
The data section of integer variables occupies two bytes in memory. Lower-order bits of this number are contained in the byte whose address is returned by the VARPTR function, and high-order bits are contained in the byte at the following address. Thus, if variable A% contains the integer 2, the address returned by the VARPTR function (the low-order byte) will contain 2, and that address plus 1 (the high-order byte) will contain 0.

	Lower byte ↓		Higher byte
2	Variable name	0	0	Data
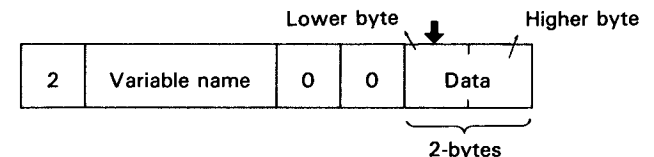
2-bytes

## (2) Single precision variables

With single precision variables, numeric values are stored in two parts, using a total of four bytes of memory. The first part which is referred to as the exponent, and the remaining three bytes are referred to as the mantissa.

The VARPTR function returns the address of the least significant byte of the mantissa; the VARPTR address + 1 contains the middle byte of the mantissa; and the VARPTR address + 2 contains the most significant byte of the mantissa. The exponent is at the VARPTR address + 3.

| 4	Variable name	0	0	Mantissa	

4-bytes / Exponent
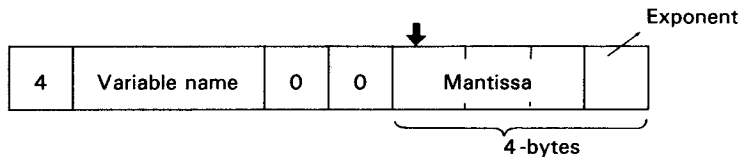
(3) Double precision variables

The storage format for double precision values in variables is the same as with single precision variables. However, the mantissa portion of a double precision variable consists of seven bytes instead of three, so the data portion of a double precision variable occupies a total of eight bytes in memory.

8	Variable name	0	0	Mantissa	

8-bytes / Exponent
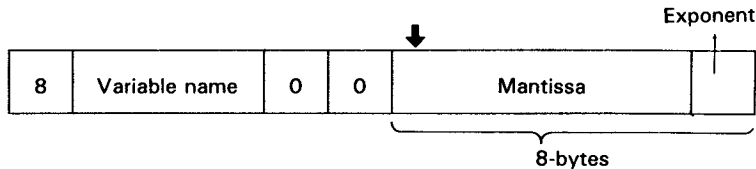
(4) String variables

With string variables, the VARPTR function returns the length of the string. The low-order byte of the string's starting address in memory is indicated by the VARPTR address + 1, and the high-order byte of the string's starting address in memory is indicated by the VARPTR address + 2.

3	Variable name	0	0	String length	Starting address

3-bytes for string descriptor
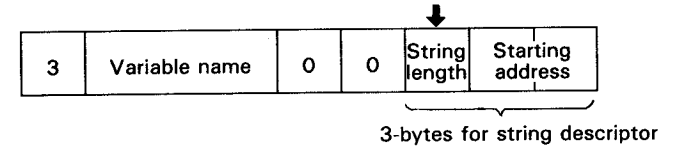
The second format returns the address of the first byte of the I/O buffer assigned to the file opened under < file number >.

This function is generally used to obtain the address of a variable prior to passing it to a machine language program. In the case of array variables, the format VARPTR(A(0)) is generally used so that the address returned is that of the lowest-numbered element of the array.

VARPTR is an abbreviation for VARiable PoinTeR.

Example

```
30 A$="abcdefghijklmnopqrstuvwxyz"
40 A=VARPTR(A$):PRINT "Address of variable A$ is ";A
50 B=PEEK(A+2)*&H100+PEEK(A+1)
60 PRINT "Address of string in variable A$ is";B
70 PRINT "String in variable A$ is ";:FOR I=0 TO 25:PRINT CH
R$(PEEK(B+I));
90 NEXT

run
Address of variable A$ is -29624
Address of string in variable A$ is 35638
String in variable A$ is abcdefghijklmnopqrstuvwxyz
Ok
```

NOTE:
The addresses of array variables change whenever a value is assigned to a new simple variable; therefore, all simple variable assignments should be made before calling VARPTR for an array.

WAIT

WAIT <port number>, J[,K]

Suspends program execution while monitoring the status of a machine input port.

The WAIT statement causes execution to be suspended until a specified machine input port develops a specified bit pattern. The data read at the port is exclusive ORed with the value of integer expression K, then ANDed with J. If the result is zero, BASIC loops back and reads the port again. If the result is not 0, execution resumes with the next statement. If K is omitted, 0 is assumed.

NOTE:
Use of this statement requires in-depth knowledge of the PX-4 firmware, and using it incautionsly can result in loss of system control and other problems.

WHILE...WEND

WHILE <expression>
    .
    .
    .
[<loop statements>]
    .
    .
    .
WEND

Allows the series of instructions between WHILE and WEND to be repeated as long as the condition specified by <expression> is satisfied.

This statement causes program execution to loop through the series of instructions between WHILE and WEND as long as the condition specified by <expression> is satisfied. <expression> is specified as any expression which has a truth value of 0 (false) or other than 0 (true). Thus numeric, logical or relational expressions may be used to specify the condition which controls looping.

As with FOR/NEXT loops, WHILE/WEND loops may be nested to any level. They may also be included within FOR/NEXT loops or vice versa. When loops are nested, the first WEND corresponds to WHILE of the innermost loop, the second WEND corresponds to WHILE of the next innermost loop, and so forth.

A WE error (WHILE without WEND) will occur if WHILE is encountered without a corresponding WEND, and a WH error (WEND without WHILE) will occur if WEND is encountered without a corresponding WHILE.

FOR...NEXT

```
10 INPUT"Enter arbitrary number";X
20 WHILE X^A<1E+06
30 PRINT STR$(X);"^";MID$(STR$(A),2,5);"=";MID$(STR$(X^A),2,
6)
40 A=A+1
50 WEND

run
Enter arbitrary number? 42.5
 42.5^0=1
 42.5^1=42.5
 42.5^2=1806.2
 42.5^3=76765.
Ok
```

WIDTH

Format **WIDTH** <no. column>,<no. lines 1>

Purpose Specifies the size of the virtual screens.

Remarks The number of columns in the virtual screen is determined by the value specified for <no. columns>. The number of lines is determined by the value specified for <no. lines>. The value specified for <no. columns> must be either 40 or 80; when 40 is specified for <no. columns>, the value specified for <no. lines> must be in the range from 8 to 50. When 80 is specified for <no. columns>, the value specified for <no. lines> must be in the range from 8 to 25.

The screen is cleared when this statement is executed.

MO error (Missing operand) — A required operand was not specified in the statement.

FC error (Illegal function call) — The number specified in one of the statement operands was outside of the prescribed range.

WIND

Format   WIND [| <counter value> |]
                    | ON
                    | OFF

Purpose   Controls forward or reverse movement of the microcassette tape and audio output to the speaker.

Remarks   When this command is executed without specifying any parameter, the tape is rewound to its beginning and the counter is reset to 0. When <counter value> is specified, the tape is wound in one direction or the other until the counter reaches the specified value. <counter value> must be specified as a numeric expression whose value lies in the range from −32768 to 32767.

Executing WIND ON places the microcassette drive in the PLAY mode and outputs the read signal to the speaker. After the drive has been placed in the PLAY mode in this manner, the BASIC interpreter goes on to execute any subsequent statements. Microcassette operation in the PLAY mode is terminated by executing WIND OFF.

The BEEP and SOUND statements cannot be executed while the microcassette drive is in the PLAY mode. Further, the WIND OFF statement is ignored if executed while the drive is not in the PLAY mode.

Note that the WIND statement cannot be executed if the tape in the drive has been installed by executing the MOUNT statement.

FC error (Illegal function call) — The value specified for <counter value> was outside the prescribed range.

OV error (Overflow) — The value specified for <counter value> was outside the prescribed range.

AC error (Tape access error) — The tape in the microcassette drive has been installed by executing the MOUNT statement.

IO error (Device I/O error) — Some problem has occurred with the microcassette drive.

WRITE

Format   WRITE[<list of expressions>]

Purpose   Displays data specified in <list of expressions> on the LCD screen.

Remarks   If <list of expressions> is omitted, a blank line is output to the LCD screen. If <list of expressions> is included, the values of the expressions are displayed on the LCD screen.

Numeric and string expressions can both be included in <list of expressions>, but each expression must be separated from the one following it with a comma. Commas are displayed between each item included, and strings displayed are enclosed in quotation marks. After the last item has been output, the cursor is automatically advanced to the next line.

The WRITE statement displays numeric values using the same format as the PRINT statement; however, no spaces are output to the left or right of numbers displayed.

See also   PRINT

WRITE

```
Enter item 1
SMITH, JOHN
Enter item 2
SMITH, ROBERT
SMITH, JOHN
SMITH, ROBERT
Ok
```

Format   WRITE # <file number>, <list of expressions>

Purpose   Used to write data to a sequential disk device file.

Remarks   <file number> is the number under which the file was opened
for output, and expressions included in <list of expressions>
are the numeric and/or string expressions which are to be writ-
ten to the file. Data is written to the file in the same format as
it is output to the screen by the WRITE statement; that is, com-
mas are inserted between individual items and strings are delimited
with quotation marks.
Therefore, it is not necessary to specify explicit delimiters in <list
of expressions>, as is the case with the PRINT # statement. The
following illustrates the difference between use of the PRINT #
and WRITE # statements (the statements indicated perform iden-
tical functions).

> **PRINT # 1,CHR$(34);"SMITH,JOHN";CHR$(34);",";**
> **CHR$(34);"SMITH, ROBERT";CHR$(34)**

> **WRITE # 1, "SMITH, JOHN","SMITH, ROBERT"**

A carriage return/line feed sequence is written to the file follow-
ing the last item in <list of expressions>.

See also   PRINT # and PRINT # USING

Example
```
10 CLS
20 OPEN "O",#1,"A:DATA"
30 FOR I=1 TO 2
40 PRINT "Enter item";I:LINE INPUT A$(I)
50 NEXT I
60 WRITE#1,A$(1),A$(2)
70 CLOSE
80 OPEN"I",#1,"A:DATA"
90 INPUT#1,A$,B$
100 PRINT A$:PRINT B$
110 CLOSE
120 END
```