BASIC 4

By: Rick Nash
Published by: Loadstar, Issue #65, 1989

Presented by: www.My64.in.nf

PREFACE

There are several BASIC extension programs available, and in my opinion, they all share a common problem. Once your program is written, it will only work if the extension program is loaded first. Worse, you can't SHARE your programs unless everyone owns the same BASIC extension that you own. It was this shortcoming that prompted me to write BASIC 4. What makes BASIC 4 different is that it attaches itself to your BASIC program, thus eliminating the problems mentioned above. Now you can use the full power of BASIC 4 in all of your programs, and freely distribute them to anyone. By the way, you won't find any programmer helper (i.e. RENUMBER, AUTO etc.) routines in BASIC 4. Every function and command is designed to help you write faster, more efficient code. I am especially proud of the array handling features. Now you can quickly search and sort string arrays, insert and delete array elements, and instantly sum an entire numeric array!

Several new commands allow greater control over your text screens. The SCREEN command will let you save or load screens to and from disk, or to and from several buffers located in RAM under ROM. The COPY, MOVE and ERASE commands offer a new dimension in screen manipulation.

I could go on raving about BASIC 4, but I think you should turn the page and discover the power for yourself.

...Rick Nash

USING BASIC 4

A very important concept to grasp early on is how BASIC 4 uses memory, and how it attaches itself to your program. A typical session will be as follows:

1. Load BASIC 4 with LOAD "BASIC 4",8 and type RUN. Once you do, you're in the development mode.

2. You will see a title screen and copyright notice. At this point you can write your code as usual, except that you have access to all of the new features of BASIC 4.

3. When you are satisfied with your code, issue a normal SAVE command to save your SOURCE code. This is more compact than the CSAVE which saves BASIC 4 along with your program. You MIGHT find it convenient to use the CSAVE option because then you can just LOAD your program and RUN it and you've booted both your program AND BASIC!

When you load A file that you've CSAVED and type list, you will only see the BASIC 4 title screen. If you type RUN however, your program will run. You'll have to break your program in order to edit it.

Steps 1-3 can be repeated as many times as necessary so that you can develop your code at different sittings (as normal BASIC allows).

See appendix A for a memory map showing actual memory used by BASIC 4.

USER GUIDE FORMAT

Each new command is listed on a separate page, and includes the following information:

1. The token for each command. Advanced programmers will find this information useful.

2. The command type. Either Function or Statement.

3. The action. A general description of the command.

4. The syntax. This section shows the syntax for proper operation. Note that parameters are enclosed in <> for clarity. Do not type these characters in your program.

5) An example. A short demonstration of the command at work.

List Of Commands

' (REM)

Token:

$CF - 207

Type:

Statement.

Action:

Shorthand notation for REMark.

Syntax:

'

Example:

```
 10 REM this is an old-fashioned remark.
 20 'here is the new style!
 30 'which do you prefer
 40 .... program continues here ....
```

See Also:
REM (In your BASIC manual).

@ (PRINT AT)

Token:
$EF - 239

Type:
Statement.

Action:

Moves cursor to desired screen location. This function is used with the PRINT command like the TAB or SPC commands. Screen locations can be expressed as ROW,COL or as a screen location 0 to 999. Like TAB and SPC, more than one @ can be used in a single print command. Multiple @'s in a single print command must be separated by a comma or semicolon.

Syntax:

```
 PRINT@<screen pos>,<variable-list>
 PRINT@(<row>, <col>),<variable-list> screen
 pos = 0-999
 row = 0-24
 col = 0-39
 var-list = normal PRINT command parameters.
```

Example:

```
10 CLS 20 PRINT@0,"THIS IS AT LOCATION 0"
30 PRINT@(10,10),"THIS IS 10,10"
40 P=780
50 PRINT@P,"780",@P+40,"820"
```

See Also:

PRINT, TAB and SPC in your BASIC manual.

ASC

Token:

```
    $F8 - 248
```

Type:

Function.

Action:

This function works exactly like the ASC in your BASIC manual except that it fixes a bug in the original. This ASC will return a 0 for a null character whereas the old version produced an error message.

Syntax:

```
ASC(<string>)
string = ascii character
```

Example:

```
 10 A$=""
 20 A = ASC(A$)
 30 PRINT "THE ASCII VALUE OF A$ IS:";A
```

See Also:

ASC in your BASIC manual.

BPEEK

Token:
$EE - 238

Type:
Function.

Action:

BPEEK (BANK PEEK) returns the value from RAM under any ROM or I/O location. This area is located from $A000 to $FFFF. BPEEK will also return the correct value from any other location, but will be slower than the normal PEEK command. This function will give you access to the normally unused RAM areas. See BPOKE for the command to poke to these areas.

Syntax:

```
 BPEEK(<memory location>)
 memory location = 0-65535
```

Example:

```
 10 CLS
 20 M=DEC("E000")
 30 PRINT@(8,0),"ENTER YOUR NAME:"
 40 A$=INLINE$(8,17,12)
 50 IFA$=""THEN 30
 60 FORI=1TOLEN(A$)
 70 BPOKEM+(I-1),ASC(MID$(A$,I,1)):NEXT
 80 CLS:FORI=1TOLEN(A$)
 90 PRINTCHR$(BPEEK(M+(I-1)));:NEXT
```

This program prompts you for your name, then pokes it into the RAM under ROM at address $E000. It then fetches your name, and displays it on the screen.

See Also:
BPOKE, PEEK and POKE.

BPOKE

Token:
$DC - 220

Type:
Statement.

Action:

The BPOKE command will poke a value to RAM under the ROM and I/O area from $A000 to $FFFF. This command will also poke to normal ram, but will be slower than the normal POKE.

Syntax:

```
 BPOKE <address>,<value>
 address = 0-65535
 value = 0-255
```

Example:

```
 10 CLS
 20 M=DEC("E000")
 30 PRINT@(8,0),"ENTER YOUR NAME:"
 40 A$=INLINE$(8,17,12)
 50 IFA$=""THEN 30
 60 FORI=1TOLEN(A$)
 70 BPOKEM+(I-1),ASC(MID$(A$,I,1)):NEXT
 80 CLS:FORI=1TOLEN(A$)
 90 PRINTCHR$(BPEEK(M+(I-1)));:NEXT
```

This program prompts you for your name, then pokes it into the RAM under ROM at address $E000. It then fetches your name, and displays it on the screen.

See Also:

BPEEK, POKE and PEEK.

CLS

Token:
 $CE - 206

Type:
 Statement.

Action:

The CLS command simply clears the screen. This is equivalent to PRINT CHR$(147).

Syntax:
 CLS

Example:

```
 10 CLS
 20 PRINT "NOTHIN' LIKE A CLEAN SCREEN!"
```

See Also:

 HOME.

COLOR

Token:
 $D7 - 215

Type:
 Statement.

Action:

The COLOR command provides an easy way to control screen border, background and character colors. Note that all three colors must be specified.

Syntax:

```
 COLOR <border>,<background>,<character>
 border = 0-15
 background = 0-15
```

character = 0-15

Example:

```
 10 A=PEEK(53280):B=PEEK(53281):C=PEEK(646)
 20 FORI=0TO15
 30 FORJ=0TO15:CLS
 40 FORK=0TO15
 50 COLOR I,J,K
 60 PRINT STRING$(40,"*");
 70 NEXT:FORL=1TO50:NEXT
 80 NEXT:NEXT
 90 CLS:COLOR A,B,C
```

This program saves the current screen colors, then displays all combination of screen and character colors. The screen is then restored to the original colors.

See Also:
 FILL

COPY

Token:
 $D3 - 211

Type:
 Statement.

Action:

The COPY command copies lines on the screen. Color memory is moved along with the characters. The original line is unchanged.

Syntax:

```
 COPY <row a> TO <row b>
 row a = source row (0-24).
 row b = destination row (0-24).
```

Example:

```
 10 CLS
 20 PRINT STRING$(40,42)
 30 PRINT@40,"*",@79,"*"
 40 FORI=2TO23:COPY1TOI:NEXT
 50 COPY 0TO24
 60 PRINT@(8,12),"A QUICK BORDER!"
 70 FORI=1TO5000:NEXT
```

See Also:

MOVE, ERASE, RVS AND FILL

CSAVE

Token:
$E6 - 230

Type:
Statement.

Action:

The CSAVE command saves the runtime BASIC module along with your source code. This combined package will LOAD and RUN like any other BASIC program. It WILL NOT list however. For this reason, YOU MUST save your "source file" while in the development mode. Use the normal SAVE command to do this. Failure to do so will result in the loss of your work. Read the manual for more information.

Syntax:
CSAVE "<filename>",<device number>,<sa> filename = the combined program filename sa = optional secondary address

Example:

```
 CSAVE "THISFILE",8
```

This command will save the current combined program as THISFILE to the disk drive number 8.

CTRL

Token:
$FC - 252

Type:
Function - system variable.

Action:

This function acts like a system variable (eg. TI$ ST etc.). It returns the current status of the SHIFT, CONTROL and COMMODORE keys. The values returned are as follows:

```
1 - shift key
2 - Commodore key
4 - control key If more than one key is
    pressed, the value returned will be
    the total of all keys pressed.
```

Syntax:
CTRL

Example:

```
 10 CLS
 20 PRINT"PRESS COMMODORE F1 TO CONTINUE..."
 30 IF (CTRL AND 2)=0THEN30
 40 IF KEY <>4 THEN 30
 50 POKE198,0
```

This program waits for the user to press the Commodore and F1 keys together before continuing the program.

See Also:
KEY

CVF

Token:
$F6 - 246

Type:
Function.

Action:

The CVF function converts a four byte string into a floating point value. The string, must have been produced by the MKF$ function. Use these functions with care if you plan on using them with disk files. Some numbers will convert to carriage returns which will mess up your sequential file.

Syntax:

```
 CVF(<string>)
 string = 4 byte string produced by MKF$
```

Example:

```
 10 CLS
 20 A=56000.678
 30 PRINT A:PRINT
 40 A$=MKF$(A)
 50 FORI=1TO4:PRINTASC(MID$(A$,I,1)):NEXT
 60 PRINT:PRINTCVF(A$)
```

This program converts a floating point value in a, into a four byte string. It then shows the contents of the string, and then converts the string back into a float again.

See Also:

CVI, MKI$ and MKF$

CVI

Token:

$F5 - 245

Type:

Function.

Action:

Converts a two byte string into an integer. The string must have been previously converted via the MKI$ function. See CVF for warnings on using the conversion commands in disk files.

Syntax:

```
    CVI(<string>)
```

Example:

```
 10 CLS
 20 PRINT "ORIGINAL",@16,+"CONVERTED"
 30 FORI=0TO10:READ D
 40 PRINT@(I+2,0),D:A$(I)=MKI$(D):NEXT
 50 FORI=0TO10
 60 PRINT@(I+2,18),CVI(A$(I)):NEXT
 70 DATA -50,2000,28765,-3897,1024,14
 80 DATA -4,32438,1798,290,2368
```

See Also:

CVF, MKI$ and MKF$

DEC

Token:

$FD - 253

Type:

Function.

Action:

Converts a hexadecimal ASCII string into a floating point value. The upper limit is approximately 2FFFFFFF. Larger numbers may work, but will be returned as scientific notation. If non-hex characters are included in the string, DEC will return the total until the first non-hex character. For example, DEC("FAXYZ") will return 250 ($FA = 250).

Syntax:

```
 DEC(<string>)
 string = valid hex ASCII characters (0-9, A-F)
```

Example:

```
 10 CLS
 20 A=DEC("E473")
 30 P=PEEK(A):A=A+1
 40 IFP=0 THEN END
 50 PRINTCHR$(P);:GOTO 30
```

See Also:
 HEX$

DEEK

Token:

 $E8 - 232

Type:

 Function.

Action:

DEEK (Double PEEK) is a 16 bit version of PEEK. It returns the 16 bit value at address and address +1. DEEK is most useful for reading system vectors. Using DEEK is the same as:
PEEK(<address>)+PEEK(<address+1>)*256

Syntax:

```
 DEEK(<address>)
 address = 0-65534
```

Example:

```
 10 CLS
 20 A=DEC("0314")
 30 READ D$:IFD$="END" THEN END
 40 PRINT D$,TAB(10)RIGHT$(HEX$(DEEK(A)),4)
 50 A=A+2:GOTO 30
 60 DATA IRQ,BRK,NMI,OPEN,CLOSE,CHKIN
 70 DATA CKOUT,CLRCH,CHRIN,CHROUT,STOP
 80 DATA GETIN,CLALL,USER,LOAD,SAVE
 90 DATA END
```

This program uses DEEK to print the low memory vectors in the Commodore 64.

See Also:

DOKE, PEEK, POKE, BPEEK and BPOKE.

DEFUSR

Token:

$D8 - 216

Type:

Statement.

Action:

The DEFUSR command sets up the USR vector at address 785-786. It is equivalent to: DOKE 785,<address>

Syntax:

```
DEFUSR(<address>)
address = address of machine language routine.
```

Example:

```
10 CLS
20 FORI=49152TO49155
30 READ D:POKEI,D:NEXT
40 DEFUSR(49152)
50 PRINT "50 TIMES 10 IS"USR(50)
60 END
70 DATA 32,226,186  :'JSR $BAE2  ;FAC1=FAC1*10
80 DATA 96          :'RTS        ;RETURN TO BASIC
```

This program sets up a small ML program that simply multiplies the number passed to it by 10. The result is returned to the BASIC program. See USR in your BASIC guide for more information.

See Also:

USR (in your basic guide).

DELETE

Token:

$E1 - 225

Type:

Statement.

Action:

The DELETE command physically removes an element from an array. All array types are supported. However, only singly-dimensioned arrays may be used in the DELETE command. After the element is deleted, the array is shifted downward from the top of the array to the deleted element. The last element is then cleared.

Syntax:

```
DELETE (array(element))
array = string, integer or float element = element        of
the array to delete
```

Example:

```
10 CLS:DIMA$(5)
20 PRINT " BEFORE"TAB(10)"AFTER":PRINT
30 FORI=0TO5:READA$(I)
40 PRINT I,A$(I):NEXT
50 DELETE(A$(3))
60 FORI=0TO5
70 PRINT@(I+2,9),I;A$(I):NEXT
80 PRINT:PRINT"NOTE THAT ELEMENT 3 HAS BEEN
   DELETED.
90 DATA CAT,DOG,TREE,APPLE,FARM,BIRD
```

At line 50 we specified that element 3 of array A$() was to be deleted. Note also that the last element (5 in this case) has been cleared.

See Also: DUP, INSERT, SCRATCH, SEARCH, SORT and SUM.

DOKE

Token:

$D0 - 208

Type:

Statement.

Action:

Pokes a 16 bit value to an address and address+1 in the standard 6502 notation (low byte, high byte). It is useful for installing vectors.

Syntax:

```
DOKE <address>,<value>
address = 0-65534
value = 0-65535
```

Example:

```
10 CLS
20 DOKE 828,49152
30 PRINT DEEK(828)
```

This example places the 16 bit value 49152 at 828 and 829. DEEK then reads and displays the 16 bit value.

See Also:

DEEK, POKE, PEEK, BPOKE and BPEEK.

DUP

Token:
$E2 - 226

Type:
Statement.

Action:
DUP is used to fill (DUPlicate) an entire array with the

same value. Any type of array with any amount of dimensions can be duplicated. Set the first element in the array to the value that you want to duplicate, then use DUP to copy it to all other elements in the array. Since string arrays are pointers to the actual text, only one string is produced, and the entire array points to it.

Syntax:

```
 DUP(<array name>(0[,0]) array name = string, integer or float
array specify element 0 (all elements should be 0 if multiply
dimensioned)
```

Example:

```
 10 DIM A$(20)
 20 DIM A%(3,3)
 30 DIMA(2,2,2)
 40 A$(0)="HELLO":DUP(A$(0))
 50 A%(0,0)=-22:DUP(A%(0,0))
 60 A(0,0,0)=176.93:DUP(A(0,0,0))
 70 CLS:FORI=0TO20:PRINTA$(I):NEXT
 80 GOSUB160
 90 CLS:FORI=0TO3:FORJ=0TO3
 100 PRINTA%(I,J),:NEXT:NEXT
 110 GOSUB160
 120 CLS:FORI=0TO2:FORJ=0TO2:FORK=0TO2
 130 PRINTA(I,J,K),:NEXT:NEXT:NEXT
 140 GOSUB160
 150 END
 160 A=PROMPT(24,10,"PRESS F1 TO CONTINUE",
    CHR$(133)):RETURN
```

See Also:
DELETE, INSERT, SCRATCH, SEARCH, SORT and SUM.

ELSE

Token:
$CD - 205

Type:
Statement.

Action:

Provides alternate action after an IF-THEN command in the case that the IF test fails. A colon must precede the ELSE command. Nested ELSE's are not supported.

Syntax:

```
 IF <expression> THEN <statement> :ELSE <statement>
    expression = test that evaluates to true or
               false.

 statement = a GOTO or line number, or other
             legal BASIC command.
```

Example:

```
 10 CLS
 20 A=50
 30 IF A=10 THEN 50:ELSE 60
 40 END
 50 PRINT"LINE 50":END
 60 PRINT"LINE 60":END
```

Since the test will fail in line 30, the program will branch to line 60.

ERASE

Token:
$D1 - 209

Type:
Statement.

Action:

The ERASE command erases a single, or range of lines on the screen.

Syntax:

```
 ERASE ERASE<row> ERASE<row a> TO <row b>

    row = 0-24
    row a = source row (0-24).
```

row b = destination row (0-24).

(The first syntax above will erase the line that the cursor is on).

Example:

```
 10 GOSUB90
 20 PRINT@120,"";:ERASE
 30 GOSUB110:GOSUB90
 40 FORI=0TO24STEP2:ERASEI:NEXT
 50 GOSUB110:GOSUB90
 60 ERASE6TO18
 70 GOSUB110:CLS
 80 END
 90 CLS:PRINTSTRING$(40,42)
 100 FORI=1TO24:COPY0TOI:NEXT
 110 FORI=1TO2000:NEXT:RETURN
```

This simple program demonstrates all three forms of the ERASE command.

See Also:

COPY, MOVE, RVS and FILL

EXEC

Token:

$FE - 254

Type:

Function.

Action:

This function may seem strange at first, but I'm sure you'll find some interesting uses for it. It will execute a string as if it were a line of BASIC code! Since this is a function, the BASIC code must return a value. Commands such as FOR-NEXT, GOTO, GOSUB and IF-THEN will not work inside of an EXEC call. Functions inside an EXEC call have access to variables in your program.

Syntax:

```
 EXEC("<command>")
 command = any normal command that returns
           a value.
```

Example:

```
 10 A$(0)="CHR$(A))
 20 A$(1)="ASC(A$))
 30 A=65
 40 A$=EXEC(A$(0))
 50 PRINT EXEC(A$(1))
```

The two strings in lines 10 and 20 are the commands that will execute. Line 30 sets variable A to 65. Line 40 executes the first string which converts the value in A into a string, and assigns it to A$. Line 50 executes the second string which converts the string value in A$ into an ASCII value which is then displayed. Whew! I told you this was strange!

See Also:
(nothing else even comes close to this one!)

FILL

Token:
$D6 - 214

Type:
Statement.

Action:

The FILL command fills color memory on the entire, or partial screen, with a specified color.

Syntax:

```
 FILL<color> FILL<color>,<row>,<col>,<# bytes>

    color = 0-15
    row = 0-24
    col = 0-39
    # bytes = 1-40
```

Example:

```
 10 LN=0:COLOR15,15,6:CLS
 20 CH$=CHR$(145)+CHR$(17)+CHR$(13)
 30 GOSUB150:FORI=0TO10
 40 PRINT@(I+4,14),"CHOICE "+RIGHT$("   "
   +MID$(STR$(I),2),2)
 50 NEXT
 60 FILL 1,LN+4,14,10
 70 ON KEY CH$; GOTO 90,100,110
 80 GOTO 70
 90 D=-1:GOTO120
 100 D=1:GOTO120
 110 PRINT@(17,8),"YOU SELECTED ITEM ";LN:END
    120 FILL 6,LN+4,14,10:LN=LN+D:IFLN<0THENLN=10
 130 IF LN>10 THEN LN=0
 140 FILL 1,LN+4,14,10:GOTO70
 150 PRINT@(23,2),"USE UP AND DOWN ARROWS TO
     CHOOSE":RVS23,0,40
 160 PRINT@(24,5),"AND PRESS RETURN TO
      SELECT";:RVS24,0,40:RETURN
```

See Also:
MOVE, ERASE, RVS and COPY

HEX$

Token:
$F2 - 242

Type:
Function.

Action:

The HEX$ function converts a floating point number to an ASCII string. The string has leading zeros, so you can extract the precision you need with the RIGHT$ command. The maximum number allowed is 2147483647 (or $7FFFFFFF).

Syntax:

```
HEX$(<number>)
```

number = any whole number 0-2147483647

Example:

```
 10 CLS
 20 PRINT"DEC   HEX":PRINT
 30 FORI=0TO15
 40 PRINTRIGHT$("  "+MID$(STR$(I),2),2);
 50 PRINTTAB(6);RIGHT$(HEX$(I),2)
 60 NEXT
```

See Also:
DEC.

HOME

Token:
$E5 - 229

Type:
Statement.

Action:

HOME places the cursor at row 0, col 0. It is the same as PRINT CHR$(19).

Syntax:

```
 HOME
```

Example:

```
 10 HOME
 20 PRINT"WELCOME HOME!"
```

This example prints a message at the HOME position.

See Also:
CLS and PRINT@.

IF

Token:
 $CC - 204

Type:
 Statement.

Action:

The IF command has been upgraded to allow the optional ELSE command. It is included in this manual because the token number has been changed. See ELSE for more information.

INLINE$

Token:
 $FB - 251

Type:
 Function.

Action:

The INLINE$ (INput LINE) function, works like the INPUT command, except that you can specify the starting position, and the maximum number of characters to accept. The only keys accepted are the ASCII characters 32-95 inclusive, the delete key, the return and stop keys.

Syntax:

```
 INLINE$(<row>,<col>,<# bytes>)
 row = 0-24
 col = 0-39
 # bytes = 1-255
```

Example:

```
 10 CLS
 20 PRINT@(8,0),"ENTER YOUR NAME:"
 30 A$=INLINE$(8,17,20)
 40 CLS:N=(40-LEN(A$))/2
 50 FORI=0TO6:PRINT@(I,N),A$;:NEXT
 60 FORI=1TO1000:FILL MOD(I,16),MOD(I,7),0,40
 70 NEXT
```

See Also:
 INPUT.

INSCR$

Token:
 $EC - 236

Type:
 Function.

Action:

The INSCR$ (INput from SCReen) command reads data directly from the screen, and places it in a string variable. The bytes are converted from screen codes to ASCII codes during the transfer.

Syntax:

```
 INSCR(<row>,<col>,<# bytes>)

     row = 0-24
     col = 0-39
     # bytes = 1-40
```

Example:

```
 10 DIMA$(23)
 20 FORI=0TO23:A$(I)=INSCR$(I,0,40):NEXT
 30 CLS:PAUSE20
 40 FORI=0TO23:PRINTA$(I);:NEXT
 50 PAUSE50
```

This program reads the screen (except the last line) into a string array. The screen is erased, and after a short delay, the screen is replaced. (See the SCREEN command for a better way to deal with screen swapping).

INSERT

Token:
 $E0 - 224

Type:
 Statement.

Action:

INSERT is used to insert a blank element at the specified subscript in an array. All singly dimensioned array types are supported. All elements from the specified subscript to the top of the array are moved up one position in the array. The top element is lost, and the specified element is cleared. Note: because the top element is always lost, make sure your array is larger than it needs to be.

Syntax:

```
 INSERT(<array>(<subscript>))

 array = String, float or integer array
```

Example:

```
 10 CLS:DIMA$(10)
 20 DATA FLOPPY DISK, COMPUTER, PRINTER
 30 DATA MODEM, SOFTWARE, BYTE
 40 FORI=0TO5:READ A$(I):NEXT
 50 PRINT"BEFORE";TAB(20);"AFTER"
 60 FORI=0TO10
 70 PRINT@(I+2,0),RIGHT$("  "+MID$(STR$(I),2),2);
 80 PRINT"  ";A$(I):NEXT
 90 INSERT(A$(2)):A$(2)=" *CHECK IT OUT!"
 100 INSERT(A$(4)):A$(4)=" *THIS IS NEW!"
 110 FORI=0TO10
 120 PRINT@(I+2,20),RIGHT$("  "+MID$(STR$(I),2),2);  130 PRINT"
";A$(I):NEXT
```

This inserts two new elements into an array and displays the new array.

See Also:
 DUP, DELETE, SCRATCH, SEARCH, SORT and SUM.

INSTR

Token:
$E7 - 231

Type:
Function.

Action:

The main string is searched to see if it contains the sub string. If it does, the position is returned, otherwise a zero is returned. An optional starting position can be specified. If it is not, then the starting position is assumed to be the first character of the main string.

Syntax:
INSTR([<pos>], <main string>, <sub string>)

pos = optional position to begin the search.
main string = the string to search.
sub string = the key to search for.

```
 10 CLS
 20 PRINT@(4,0),"PHONE:":Z$="(...) ...-...."
 30 ROW=4:COL=7:LN=14
 40 GOSUB 500:CLS:PRINT"YOU ENTERED "A$:END
 500 FLAG=0:CT=0:P=0:PRINT@(ROW,COL),Z$
 510 A=ASC(INSCR$(ROW,COL+P,1)):IF((P<LN)AND
    (A<>46))THENP=P+1:GOTO510
 520 CT=CT+1:IFCT=20THENFLAG=XOR(FLAG,1):CT=0:RVS
ROW,COL+P,1
 530 GETA$:IFA$=""THEN520
 540 IFINSTR("1234567890",A$)THEN 570
 550 ONINSTR(CHR$(13)+CHR$(20),A$) GOTO 600,610
 560 GOTO 520
 570 IFP=LNTHEN 510
 580 IFFLAG=1THENRVS ROW,COL+P,1
 590 PRINT@(ROW,COL+P),A$;:P=P+1:GOTO510
 600 A$=INSCR$(ROW,COL,LN):RETURN
 610 PRINT@(ROW,COL+P)," ";:GOTO500
```

This example prompts the user to enter a phone number. It uses INSTR to accept only the number keys, or <RETURN> and <DEL>.

KEY

Token:
 $F7 - 247

Type:
 Function - system variable.

Action:

This function acts like a system variable (i.e. TI$, ST etc.). It returns the scan code (not ASCII) of the current key being pressed. This function is the same as PEEK(203). See appendix B for a list of scan codes. The ON command has been modified to recognize the KEY function. See ON for more information.

Syntax: KEY

Example:

```
 10 CLS
 20 PRINT "PRESS COMMODORE F1 TO CONTINUE..."
 30 IF (CTRL AND 2)=0THEN30
 40 IF KEY <> 4 THEN 30
 50 POKE198,0
```

See Also:
 CTRL and ON.

MKF$

Token:
 $F4 - 244

Type:
 Function.

Action:

The MKF$ function converts a floating point value, into a 4 byte string. See CVF for warnings on using these strings in disk files.

Syntax:

```
 MKF$(<float>)
 float = any floating point value.
```

Example:

```
 10 CLS
 20 A = 56000.678
 30 PRINT A:PRINT
 40 A$=MKF$(A)
 50 FORI=1TO4:PRINTASC(MID$(A$,I,1)):NEXT
 60 PRINT:PRINTCVF(A$)
```

See Also:

CVF, MKI$ and CVI.

MKI$

Token:

$F3 - 243

Type:

Function.

Action:

The MKI$ converts any integer into a two byte string. See CVF for warnings on using these strings in disk files.

Syntax:

```
 MKI$(<integer>)
```

Example:

```
 10 CLS:DIMA$(10)
 20 PRINT"ORIGINAL",@16,"CONVERTED"
 30 FORI=0TO10:READ D
 40 PRINT@(I+2,0),D:A$(I)=MKI$(D):NEXT
 50 FORI=0TO10
 60 PRINT@(I+2,18),CVI(A$(I)):NEXT
 70 DATA -50,2000,28765,-3897,1024,14
 80 DATA -4,32438,1798,290,2368
```

See Also:

MKF$, CVI and CVF.

MOD

Token:
$F1 - 241

Type:
Function.

Action:

The MOD function returns the remainder of an integer division.

Syntax:

```
MOD(<integer a>, <integer b>)
```

integer a = dividend.
integer b = divisor.

Example:

```
10 CLS
20 PRINT "THE REMAINDER OF 10 / 4 IS" MOD(10,4)
```

See Also:
QUOT.

MOVE

Token:
$D4 - 212

Type:
Statement.

Action:

MOVE copies a screen row to another screen row. It then clears the original row.

Syntax:

```
 MOVE <row a> TO <row b>

 row a = source row (0-24).
 row b = destination row (o-24).
```

Example:

```
 10 CLS
 20 PRINT STRING$(40,"*");
 30 PRINT "**    MOVIN' RIGHT ALONG  **";
 40 PRINT STRING$(40,"*");:PAUSE10
 50 FORI=0TO21:MOVEI+2TOI+3:MOVEI+1TOI+2:
    MOVEITOI+1:NEXT
 60 FORI=24TO3STEP-1:MOVEI-2TOI-3:MOVEI-1TO
    I-2:MOVEITOI-1:NEXT
 70 PRINT@(5,3),"HOW'S THAT FOR A MOVING MESSAGE"
```

This example prints a "moving message". Note that the string in line 30 is 40 characters wide.

See Also:

COPY, ERASE, RVS and FILL.

ON

Token:
$DE - 222

Type:
Statement.

Action:

The ON command has been upgraded to work with the KEY command. The ON command works as before, but now you can also test for keystrokes. Please note the use of the semicolon in the syntax.

Syntax:

```
ON KEY <string> ; GOSUB / GOTO <linenumber>[,<linenumber>]...
```

string = ASCII keys to match.

Example:

```
 10 CLS:PRINT"PRESS A-D:"
 20 ON KEY "ABCD";GOTO 100,200,300,400
 30 GOTO 20
 100 PRINT"YOU PRESSED A":END
 200 PRINT"YOU PRESSED B":END
 300 PRINT"YOU PRESSED C":END
 400 PRINT"YOU PRESSED D":END
```

The example above waits for a key A-D (as specified in the literal string). When one is pressed, a message indicates which key it was. Note the semi- colon in line 20. You will get a syntax error without it.

PAUSE

Token:
$D5 - 213

Type:
Statement.

Action:

The PAUSE command causes a delay. An optional number specifies the duration (in 1/10 second increments). If no number is given, then the delay will continue until a the <RETURN> key is pressed. Note that the <STOP> key is scanned during the delay, so you can abort long delays. The 1/10 figure is approximate.

Syntax:

```
 PAUSE [<num>]
```

num = optional number of 1/10 seconds in the delay.

Example:

```
 10 CLS
 20 PRINT "A 10 SECOND DELAY..."
 30 PAUSE120
 40 PRINT "PAUSE UNTIL <RETURN> IS PRESSED..."
 50 PAUSE
```

PDELAY

Token:
 $D9 - 217

Type:
    Statement.

Action:

    The PDELAY command sets the blink rate for the PROMPT command. If a PDELAY of 0 is specified, then the PROMPT command will not blink.

Syntax:

```
    PDELAY <num>

    num = blink rate (0-255).
    0 = no blink.
```

Example:

```
    10 CLS
    20 PDELAY 15
    30 A=PROMPT(8,10,"CONTINUE (Y/N)","YN")
    40 IF CHR$(A)="N"THEN30
```

See Also:
    PROMPT.

PROMPT

Token:
    $ED - 237

Type:
    Function.

Action:

The PROMPT command will display a message at a specified location on the screen. PROMPT then waits for a key press that matches one of the characters in the validation string. Once a valid key is pressed, its ASCII value is returned by the PROMPT function.

Syntax:

```
 PROMPT(<row>,<col>,<message>,<validation string>)
```

    row = 0-24
    col = 0-39
    message = message to display
    validation string = ASCII keys that are allowed, to cause the program to resume.

Example:

```
    10 CLS
    20 PDELAY 15
    30 A=PROMPT(8,10,"CONTINUE (Y/N)","YN")
    40 IF CHR$(A)="N"THEN30
```

See Also:
PDELAY.

QUOT

Token:
$F0 - 240

Type:
Function.

Action:

The QUOT function returns the quotient from an integer division. The MOD function can be used to return the remainder.

Syntax:

```
    QUOT(<num a>, <num b>)
    num a = dividend
    num b = divisor
```

Example:

```
    10 CLS
    20 PRINT "100 DIVIDED BY 6 IS";QUOT(100,6)
    30 PRINT
    40 PRINT "WITH A REMAINDER OF";MOD(100,6)
```

See Also:
    MOD.

RESTORE

Token:
    $E4 - 228

Type:
    Statement.

Action:

The RESTORE works like the normal BASIC version, except that you can specify a line number to restore to. This feature allows you to access DATA statements in any order that you wish.

Syntax:

```
    RESTORE [<line number>]
    line number = optional line to set DATA
      pointer to.
```

Example:

```
    10 DATA SPECIFY THE LINE
    20 DATA DATA POINTER TO BE SET TO.
    30 DATA NOW YOU CAN
    40 DATA THAT YOU WANT THE
    50 CLS
    60 RESTORE30:READD$:PRINTD$
    70 RESTORE10:READD$:PRINTD$
    80 RESTORE40:READD$:PRINTD$
    90 RESTORE20:READD$:PRINTD$
```

See Also:

DATA and RESTORE in your BASIC manual.

RVS

Token:

    $D2 - 210

Type:

    Statement.

Action:

The RVS command will invert the characters on a specified area of the screen.

Syntax:

```
RVS <row>,<col>,<# chars>
row = 0-24
col = 0-39
# chars = 1-40
```

Example:

```
10 LN=0:COLOR15,15,6:CLS
20 CH$=CHR$(145)+CHR$(17)+CHR$(13)
30 GOSUB150:FORI=0TO10
40 PRINT@(I+4,14),"CHOICE "+RIGHT$(
  "  "+MID$(STR$(I),2),2)
50 NEXT
60 RVS LN+4,13,11
70 ON KEY CH$; GOTO 90,100,110
80 GOTO 70
90 D=-1:GOTO120
100 D=1:GOTO120
110 PRINT@(17,8),"YOU SELECTED ITEM ";LN:END
120 RVS LN+4,13,11:LN=LN+D:IFLN<0THENLN=10
130 IF LN>10 THEN LN=0
140 RVS LN+4,13,11:GOTO70
150 PRINT@(23,2),"USE UP AND DOWN ARROWS TO
   CHOOSE":RVS23,0,40
160 PRINT@(24,5),"AND PRESS RETURN TO
   SELECT";:RVS24,0,40:RETURN
```

This example uses the RVS command to make a nice "scrolling bar" menu.

See Also:

ERASE, MOVE, FILL and COPY.

SCRATCH

Token:
    $E3 - 227

Type:
    Statement.

Action:

The SCRATCH command is used to delete an entire array, and return the memory back to the system. Think of the SCRATCH command as kind of an UN-DIM. If the array is of type string, then all strings are released from the string table.

Syntax:

```
SCRATCH(<array name>(0))
```

array name = string, float or integer array.

Example:

```
10 CLS:PRINT"FREE MEMORY-":PRINT
20 PRINT"   BEFORE DIM:" 65535-FRE(0)
30 DIMA(200)
40 PRINT"    AFTER DIM:"65535-FRE(0)
50 SCRATCH(A(0))
60 PRINT"AFTER SCRATCH:" 65535-FRE(0)
70 PRINT:PRINT "NOTE THAT ALL MEMORY HAS BEEN
   RETURNED."
```

See Also:

DELETE, DUP, INSERT, SEARCH, SORT and SUM.

SCREEN

Token:
    $DD - 221

Type:
    Statement.

Action:

The SCREEN command is used to SAVE and LOAD text screens to/from disk. Screens can also be saved and loaded from one of four buffers under the KERNAL ROM. When using the buffers, two operations can be performed. Exchange will swap the two screens. Put will copy the source screen to the destination.

Syntax:

```
    SCREEN(<operation>, <source>, <dest>)

      operation = E for exchange, P for put.
      source = 0-4 (display is 0, buffers are 1-4)
      dest = 0-4 (display is 0, buffers are 1-4)


    SCREEN(<operation>,<num>,<filename>)

      operation = S for save to disk, L for load.
      num = source number for save, destination for load.
      filename = any legal disk file name.
```

Note: num can be omitted from the disk load version. In that case, the screen is put into the same buffer number from which it was saved.

Example:

```
    10 SCROFF:FORI=1TO4:CLS
    20 PRINT@(I,8),"THIS IS SCREEN" I
    25 PRINT:FORJ=1TO18:PRINTSTRING$(40,64+J);:NEXT
    30 FILLI-1:SCREEN(P,0,I):NEXT:CLS:SCRON
    40 FORI=1TO4:SCREEN(P,I,0):PAUSE60:NEXT
```

This example stashes away 4 screens, then displays.

SCROFF

Token:
    $DB - 219

Type:
    Statement.

Action:

The SCROFF command turns off the video display. This is useful for drawing screens without the user being able to see them being drawn. Care should be taken so that errors do not happen during a SCROFF, if they do, the error message will not be seen!. Press RUN-STOP/RESTORE to restore normal video if this happens.

Syntax:
    SCROFF

Example:

```
    10 SCROFF
    20 CLS
    30 PRINTSTRING$(80,"*");
    40 FORI=1TO14:PRINT"**"SPC(36)"**";:NEXT
    50 PRINTSTRING$(80,"*");
    60 PRINT@(8,6),"THIS WILL APPEAR INSTANTLY!"
    70 SCRON:PAUSE120
```

See Also:
    SCRON.

SCRON

Token:
    $DA - 218

Type:
    Statement.

Action:

The SCRON command turns on the screen after SCROFF had been used to turn it off.

Syntax:
    SCRON

Example:

```
    10 SCROFF
    20 CLS
    30 PRINTSTRING$(80,"*");
    40 FORI=1TO14:PRINT"**"SPC(36)"**";:NEXT
    50 PRINTSTRING$(80,"*");
    60 PRINT@(8,6),"THIS WILL APPEAR INSTANTLY!"
    70 SCRON:PAUSE120
```

See Also:

    SCROFF.

SEARCH

Token:
    $F9 - 249

Type:
    Function.

Action:

The SEARCH command is used to quickly search a string array for a specified search key. The array can be searched in any one of six different relational operations. If the key is found, then SEARCH returns the element number of the match. If the key is not found, then SEARCH returns –1.

Syntax:

```
    SEARCH(<operator>, <array$(0)>, <key>)

  operator: 1 = less than
            2 = equal to
            3 = less than or equal to
            4 = greater than
            5 = not equal to
            6 = greater than or equal to
  array = string array to be searched.
  key = search key.
```

Example:

```
10 CLS:DIMA$(5):FORI=0TO5:READA$(I):NEXT
20 DATA ZEBRA,CAR,COMPUTER,RADIO,APPLE,TREE
30 FORI=0TO5:PRINTI,A$(I):NEXT
40 PRINT:PRINT
50 K$="TREE":GOSUB100
60 K$="CAR":GOSUB100
70 K$="AUTO":GOSUB100
80 END
100 S=SEARCH(2,A$(0),K$)
110 IFS>-1THENPRINTK$" WAS FOUND AT ELEMENT "S
120 IFS<0THENPRINTK$" WAS NOT FOUND"
130 RETURN
```

See Also:

INSERT, DELETE, DUP, SCRATCH, SORT and SUM.

SORT

Token:

$DF - 223

Type:

Statement.

Action:

The SORT command is used to sort a string array into ascending or descending order. It uses the Shell Metzner sorting algorithm. Note that element zero is not sorted.

Syntax:

```
SORT(<direction>, <array$(0)>)

direction = A for ascending.
            D for descending
array = string array to sort.
```

Example:

```
10 CLS:DIMA$(6):FORI=1TO6:READA$(I):NEXT
20 DATA RADIO,ZEBRA,COMPUTER,CAR,APPLE,TREE
```

```
30 PRINT"UNSORTED","ASCENDING","DESCENDING":
  PRINT
40 FORI=1TO6:PRINTA$(I):NEXT
50 SORT(A,A$(0)):PRINT@(2,0),"";
60 FORI=1TO6:PRINT ,A$(I):NEXT
70 SORT(D,A$(0)):PRINT@(2,0),"";
80 FORI=1TO6:PRINT ,,A$(I):NEXT
```

This simple program demonstrates the SORT function. A small array is loaded with strings, it is then sorted in ascending and descending order and displayed on the screen

See Also:

INSERT, DELETE, DUP, SCRATCH, SEARCH and SUM.

STRING$

Token:

$E9 - 233

Type:

Function.

Action:

The STRING$ function returns a string of n copies of the specified character (up to 255).

Syntax:

```
STRING$(<num>,<string>)

  num = number of copies
  string = the character to copy

STRING$(<num>,<ASCII number>)

  num = number of copies
  ASCII number = ASCII value of desired
                 character.
```

Example:

```
10 CLS:A$=CHR$(45)
20 PRINT STRING$(40,"-")
```

```
30 PRINT STRING$(40,45)
40 PRINT STRING$(40,A$)
```

This program demonstrates the several ways of passing the string parameter to the STRING$ command.

SUM

Token:

$FA - 250

Type:

Function.

Action:

The SUM function returns the sum of an entire numeric array. The array must be floating point or integer.

Syntax:

```
SUM(<array>(0))
```

array = an integer or float array.

Example:

```
10 CLS:DIM A(10)
20 DATA 500,299.60,53.80,40,20,1000,
  67.3,666.23,123.48,87,200
30 FORI=0TO10:READ A(I):NEXT
40 PRINT"THE SUM OF:"
50 FORI=0TO10:PRINTTAB(10)A(I):NEXT
60 PRINTTAB(10)"----------"
70 PRINT  TAB(6)"IS: "SUM(A(0))
```

See Also:

INSERT, DELETE, DUP, SCRATCH, SEARCH and SORT.

VARPTR

Token:
    $EA - 234

Type:
    Function.

Action:

The VARPTR function returns the address of the specified variable. Note that strings return a pointer to the string, and its length. For more information on variables and how they are stored in memory, see TOOL KIT BASIC, by Dan Heeb, published by COMPUTE! BOOKS, or MASTERING THE COMMODORE 64 by Jones & Carpenter, published by WILEY PRESS.

Syntax:

```
VARPTR(<variable>)
```

variable = any legal BASIC variable.

Example:

```
10 CLS
20 DIM A$,A,B,I
30 A$="HERE IS A STRING!"
40 A = VARPTR(A$)
50 B = DEEK(A+1)
60 FORI=1TOPEEK(A):PRINTCHR$(PEEK(B+I-1));:
   NEXT
```

This example uses VARPTR to locate a string variable. The string is then printed on the screen.

XOR

Token:
    $EB - 235

Type:
    Function.

Action:

The XOR function performs the bitwise exclusive-or operation. Like AND or OR, XOR works on individual bits of a byte. The following truth table explains:

First Bit	Second Bit	Result
0	0	0
0	1	1
1	0	1
1	1	1

The XOR function is useful for flipping between two characters, or flag conditions.

Syntax:

```
XOR(<value>, <value>)
value = 0-32767
```

Example:

```
10 CLS:F1=0:F2=0:A$(0)="ON ":A$(1)="OFF"
20 PRINT"YOU TYPE, AND I'LL PRINT THE CHARACTERS"
30 PRINT"ON THE SCREEN. IF YOU PRESS THE * KEY"
40 PRINT"I WON'T SHOW ANY CHARACTERS UNTIL YOU"
50 PRINT"PRESS THE * KEY AGAIN."
60 GETA$:IFA$=""THEN60
70 IFF1=0THENCLS:PRINT@36,A$(F2):F1=1
80 IFA$<>"*"THEN100:ELSE F2=XOR(F2,1):
  POKE783,1:SYS65520
90 PRINT@36,A$(F2):POKE783,0:SYS65520:GOTO60
100 IFF2=1THEN60:ELSE PRINTA$;:GOTO 60
```

See Also:
    AND, OR and NOT in your BASIC manual.

APPENDIX A

MEMORY MAP

```
00000 - $0000 = Start of RAM
02049 - $0801 = Start of Runtime module
06400 - $1900 = Approximate new start of BASIC
65535 - $FFFF = Top of RAM
```

APPENDIX B

SCAN CODES

KEY	CODE	KEY	CODE
INSERT/DELETE	0	9	32
RETURN	1	I	33
CURSOR RIGHT	2	J	34
F7	3	0	35
F1	4	M	36
F3	5	K	37
F5	6	O	38
CURSOR DOWN	7	N	39
3	8	+	40
W	9	P	41
A	10	L	42
4	11	-	43
Z	12	.	44
S	13	:	45
E	14	@	46
(NOT USED)	15	,	47
5	16	BRITISH PND	48
R	17	*	49
D	18	;	50
6	19	CLEAR/HOME	51
C	20	(NOT USED)	52
F	21	=	53
T	22	UP ARROW	54
X	23	/	55
7	24	1	56
Y	25	BACK ARROW	57
G	26	(NOT USED)	58
8	27	2	59
B	28	SPACE BAR	60
H	29	(NOT USED)	61
U	30	Q	62
V	31	RUN/STOP	63
NO KEY PRESSED	64		

ADDENDUM

RASTER SYNC

BASIC 4 synchronizes certain commands with the raster line on the screen. What this means is that FILL, RVS, COPY, MOVE, SCREEN, etc. won't write to the screen while it's still being updated. This makes screen manipulation look smoother at the expense of printing to the screen seemingly slower.

If you're constantly using BASIC 4 commands that manipulate the screen, you might notice the decrease in speed. It might sound hard to believe but those waits of up to a max 30th of a second can add up to to notable intervals when nested in busy screen manipulation loops. If this isn't acceptable, you can disable the wait. To disable the raster wait, put this command at the top of your program:

```
POKE 823,0:REM DISABLE RASTER WAIT
```

Likewise you can enable the raster wait with the following:

```
POKE823,255:REM ENABLE RASTER WAIT
```

NEW BUT COMPATIBLE SYNTAX FOR INLINE$

Syntax:

```
    INLINE$(<row>,<col>,<#bytes>,[<validation string])
```

The INLINE$ command now accepts upper/lowercase letters and numbers as a default. You can also specify which characters are acceptable through an OPTIONAL validation string which can be specified in quotes or through a string variable. For instance:

```
    INLINE$(20,14,10,"1234567890.-")
```

This will place a cursor at row 20, column 14 and allow only 10 characters to be entered. These ten characters are specified in the validation string, "1234567890.-". Only these characters will be accepted.

A null or absent validation string paramter will allow all the default characters to be entered.

The maximum number of characters allowed is 81.

If you'd like a different type cursor to flash while INLINE$ is in use, POKE the ASCII value of the character you'd like to flash in location 822.

The maximum length of a validation string is 127.

DISABLING BASIC 4

You can disable BASIC with a simple SYS 58451.

This will make everything normal except the start of BASIC. Every command used after this MUST be BASIC V2.

Note: THAT THE START OF BASIC MUST BE MANUALLY MOVED BACK TO $0801.

SYS2214 will do a warm start which will reset the computer without killing fastloads. BASIC pointers will be normal but your program can't continue after this point since it will be NEWed. You are left in the immediate mode.