Radio Shack

TRS-80
Pocket
Handbook

U.S.A. $6.95

TRS-80 Pocket BASIC Handbook

by
William Barden, Jr.

Radio Shack
A Tandy Corporation

Second Edition
First Printing-1983

Preface

This book is designed to be a quick reference guide to the BASIC languages used on the Radio Shack TRS-80 Model I, Model II, Model III, and Color Computers. It won't replace the BASIC manuals that come with those computer systems, but·it will help to jog your memory about the types of BASIC commands available, the format of the commands, the operation of the commands, and the commands that are related.

The commands include all BASIC symbols, such as * for multiply, all BASIC commands, such as PRINT, and all BASIC functions, such as ATN. We'll use the generic term "command" to mean any of those three items. The term "statement" will be any use of commands in a single step, such as A=SIN(B*C) or POKE 16523,(RR/67). The term BASIC "line" will mean a single statement or multiple statements with the same line number.

There are 255 commands in this reference book, one per page. They are organized in alphabetical order. The Contents section on the next few pages lists all commands and indicates for which systems they are used. The systems are: Model I, Level I; Model I, Level II; Model I, Disk BASIC; Model II BASIC; Model III, Level I; Model III, Level III; Model III, Disk BASIC, Color Computer, BASIC; Color Computer Extended BASIC; and Color Computer, Disk BASIC. We'll keep this order in the SYSTEM description on each page.

Each command format is described under "FORMAT". In those cases where the command is normally used in a program we've included "line#" under the format. In those cases where the command is normally used in the command mode, we've left out the "line#". In some cases the command is used in either command mode or program execution, and we've indicated both by two format statements, one with "line#" and one without.

In those cases where a command requires parentheses or double quotes, we've included them in the FORMAT. Dots indicate that the command may be embedded in other commands and probably won't stand by itself, as in the case of functions.

The EXAMPLES show one or more actual examples of the use of the command. Descriptive text is sometimes included in lower case in the right-hand portion of the examples.

The DESCRIPTION section contains a very brief explanation of the command. Any peculiarities for specific systems are also described here.

RELATED COMMANDS lists any commands that may help in understanding the action of the command in question.

To Babbage for starting the whole thing!

Contents

CMD"A"	......o...
CMD"B"	......o...
CMD"C"	......o...
CMD"D"(I)	..o.......
CMD"D"(III)	......o...
CMD"E"	......o...
CMD"I"	......o...
CMD"J"	......o...
CMD"L"	......o...
CMD"O"	......o...
CMD"P"	......o...
CMD"R"	..o...o...
CMD"S"	..o...o...
CMD"T"	..o...o...
CMD"X"	......o...
CMD"Z"	......o...
COLOR	........oo
CONT	oooooooooo
COPY	.........o
COS	.ooo.oo.oo
CSAVE	ooo.oooooo
CSAVE#-	.oo.......
CSAVEM	........oo
CSNG	.ooo.oo...
CVD	..oo..o...
CVI	..oo..o...
CVN	.........o
CVS	..oo..o...
D	.ooo.oo...
DATA	oooooooooo
DATE$	...o......
DEF FN	..oo..o.oo
DEFDBL	.ooo.oo...
DEFINT	.ooo.oo...
DEFSNG	.ooo.oo...
DEFSTR	.ooo.oo...
DEFUSR	..oo..o.oo
DEL	........oo
DELETE	.ooo.oo...
DIM	.ooo.ooooo
DIR	.........o
DRAW	........oo
DRIVE	.........o
DSKI$	.........o
DSKINI	.........o
DSKO$	.........o
E	.ooo.oo...
EDIT	.ooo.oo.oo
Edit Mode A	.ooo.oo.oo
Edit Mode Backspace	.ooo.oo.oo
Edit Mode C	.ooo.oo.oo
Edit Mode D	.ooo.oo.oo
Edit Mode E	.ooo.oo.oo
Edit Mode ENTER	.ooo.oo.oo
Edit Mode ESC	...o......
Edit Mode H	.ooo.oo.oo
Edit Mode I	.ooo.oo.oo
Edit Mode K	.ooo.oo.oo
Edit Mode L	.ooo.oo.oo
Edit Mode Q	.ooo.oo.oo
Edit Mode S	.ooo.oo.oo
Edit Mode SHIFT, up arrow	.ooo.oo.oo
Edit Mode Space-Bar	.ooo.oo.oo
Edit Mode X	.ooo.oo.oo
END	oooooooooo
EOF	..oo..o..o
EQV	...o......
ERASE	...o......
ERL	.ooo.oo...
ERR	.ooo.oo...
ERROR	.ooo.oo...
ERR$	...o......
EXEC	.......ooo
EXP	.ooo.oo.oo
FIELD	..oo..o..o
FILES	.........o
FIX	.ooo.oo.oo
FOR...TO...STEP	oooooooooo
FRE	.ooo.oo...
FREE	.........o
GET(disk)	..oo..o..o
GET(graphics)	........oo
GOSUB	oooooooooo
GOTO	oooooooooo
HEX$	...o....oo
IF...THEN	oooooooooo
IF...THEN...ELSE	.ooo.ooooo
IMP	...o......
INKEY$	.ooo.ooooo
INP	.oo..oo...
INPUT	oooooooooo
INPUT...;	oooooooooo
INPUT#(disk)	..oo..o..o

INPUT#(non-disk)	o.........
INPUT#-1	.oo..ooooo
INPUT#-2	.oo.......
INPUT$(disk)	...o......
INPUT$(non-disk)	...o......
INSTR	..oo..o.oo
INT	oooooooooo
JOYSTK	.......ooo
KILL	..oo..o..o
LEFT$	.ooo.ooooo
LEN	.ooo.ooooo
LET	oooooooooo
LINE	........oo
LINE INPUT	..oo..o.oo
LINE INPUT#	..oo..o..o
LIST	oooooooooo
LLIST	.ooooooooo
LOAD	..oo..o..o
LOADM	.........o
LOC	...o..o..o
LOF	..oo..o..o
LOG	.ooo.oo.oo
LPRINT	.oooooo...
LPRINT USING	.ooo.oo...
LSET	..oo..o..o
MEM	oooooooooo
MERGE	..oo..o..o
MID$	.ooo.ooooo
MID$=	..oo..o.oo
MKD$	..oo..o...
MKI$	..oo..o...
MKN$	.........o
MKS$	..oo..o...
MOD	...o......
MOTOR	.......ooo
NAME(renumber)	......o...
NEW	oooooooooo
NEXT	oooooooooo
NOT	.ooo.ooooo
OCT$	...o......
ON ERROR GOTO	.ooo.oo...
ON...GOSUB	oooooooooo
ON...GOTO	oooooooooo
OPEN	..oo..oo.o
OR	.ooo.ooooo
OUT	.oo..oo...
PAINT	........oo
PCLEAR	........oo
PCLS	........oo
PCOPY	........oo
PEEK	.oo..ooooo
PLAY	........oo
PMODE	........oo
POINT	ooo.oooooo
POKE	.oo..ooooo
POS	.ooo.oo.oo
PPOINT	........oo
PRESET	........oo
PRINT	oooooooooo
PRINT#-1	.oo..ooooo
PRINT#-2(CC)	.......ooo
PRINT#-2(I)	.oo.......
PRINT#(disk)	..oo..o..o
PRINT#(non-disk)	o.........
PRINT USING	.ooo.oo.oo
PRINT AT	o...o.....
PRINT @	.ooo.ooooo
PSET	........oo
PUT(disk)	..oo..o..o
PUT(graphics)	........oo
RANDOM	.ooo.oo...
READ	oooooooooo
REM	oooooooooo
RENAME(CC)	.........o
RENUM	...o....oo
RESET	ooo.oooooo
RESTORE	oooooooooo
RESUME	.ooo.oo...
RETURN	oooooooooo
RIGHT$	.ooo.ooooo
RND	oooooooooo
ROW	...o......
RSET	..oo..o..o
RUN	oooooooooo
RUN"prog"	..oo..o..o
SAVE	..oo..o..o
SAVEM	.........o
SCREEN	........oo
SET	ooo.oooooo
SGN	.ooo.ooooo
SIN	.ooo.ooooo
SKIPF	.......ooo
SOUND	.......ooo
SPACE$	...o......

SPC	...o......
SQR	.ooo.oo.oo
STOP	oooooooooo
STR$	.ooo.ooooo
STRING$	.ooo.ooooo
SWAP	...o......
SYSTEM(I/III)	.oo..oo...
SYSTEM(II)	...o......
TAB	oooooooooo
TAN	.ooo.oo.oo
TIME$	..oo.oo...
TIMER	........oo
TROFF	.ooo.oo.oo
TRON	.ooo.oo.oo
UNLOAD	.........o
USR	.ooo.ooooo
USRn	..oo..o.oo
VAL	.ooo.ooooo
VARPTR	.ooo.oo.oo
VERIFY	.........o
WRITE#	.........o
XOR	...o......
up arrow	oooooooooo
reverse slash	...o......

SPECIAL KEYS
ERROR CODES
ASCII CODES
BINARY, DECIMAL, HEXADECIMAL EQUIVALENTS

```
                          oooooooooo
Configuration = I, LVL I
                I, LVL II
                I, Disk
                II
                III, LVL I
                III, LVL III
                III, Disk
                CC, BASIC
                CC, Ext BASIC
                CC, Disk
```

COMMANDS

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...variable name ! . . .

EXAMPLES

```
1000 A!=123456
1010 ZZ!=99999
```

DESCRIPTION

The suffix "!" is used to define single-precision variables. The default variable type is single precision, but the "!" suffix can be used to define a variable within a range used on a DEFDBL, DEFINT, or DEFSTR. Single-precision variables hold 7 decimal digits of precision in memory and display 6 decimal digits. Single-precision variables take up four bytes of RAM storage for each variable.

RELATED COMMANDS

DEFDBL, DEFINT, DEFSNG, DEFSTR

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#..."string literal"...

EXAMPLES

```
1000 A$=‘‘THIS IS A STRING’’
```

DESCRIPTION

Double quotes are used to enclose string "literals". String literals are the actual text of the string. They are stored in the BASIC program line itself, although they may be used to create new strings that are stored in the string storage area. String literals may generally be used any time that a string variable can be used, such as in PRINT statements, string comparisons, or other string processing. Always enclose the string literal with double quotes; failure to do so may cause errors in program renumbering or other program processing.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...variable name#...

EXAMPLES

```
1000 A#=1234567890.1234567
1010 ZZ#=99999999999
```

DESCRIPTION

The suffix "#" is used to define double-precision variables. The default variable type is single precision. Other numeric variable types must be defined by the %, #, D, or $ suffixes, or by DEFINT, DEFDBL, or DEFSTR. The "#" suffix can be used to define a double-precision variable within a range used on a DEFINT, DEFDBL, or DEFSTR. Double-precision variables hold 17 decimal digits of precision in memory and display 16 decimal digits. Double-precision variables take up eight bytes of RAM storage for each variable. Double-precision variables should be used in place of single-precision variables where extreme accuracy is desired and when the number of double-precision variables will not be prohibitively large (as would be the case in a large array).

RELATED COMMANDS

DEFDBL, DEFINT, DEFSNG, DEFSTR

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...variable name$...

EXAMPLES

```
1000 A$=''TELEPHONE #''
1010 ZZ$=STRING$(100,''*'')
```

DESCRIPTION

The suffix "$" is used to define string variables. String variables generally hold ASCII character data, although they may hold other non-ASCII data as well. String variables may be from 0 to 255 characters long, where each character corresponds to one byte in RAM. The names of string variables follow the same rules for numeric variable names. The first character must be alphabetic. One or two character names may be used. (Model I/III Level I allows only A$ and B$.) The suffix "$" denotes the variable as a string variable; the same name may be used for a numeric and string variable, except that the suffix will be different. AA$ and AA are a string variable and numeric variable, respectively. The suffix "$" may be used to define a string variable within a range of other variables defined by a DEFDBL, DEFSNG, or DEFINT.

RELATED COMMANDS

DEFDBL, DEFINT, DEFSNG, DEFSTR

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...variable name%...

EXAMPLES

```
1000 A%=-12345
1010 ZZ%=9999
```

DESCRIPTION

The suffix "%" is used to define integer variables. The default variable type is single precision, but the "%" suffix can be used to define an integer variable explicitly or within a range used on a DEFDBL, DEFSNG, or DEFSTR. Integer variables hold values from -32768 through +32767. No fractions are allowed. Integer variables take up two bytes of RAM storage for each variable, making them one of the most efficient ways to store data, when the data is in the limited range of values.

RELATED COMMANDS

DEFDBL, DEFINT, DEFSNG, DEFSTR

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...&Hdddd...

EXAMPLES

```
1010 FOR I=&H8000 TO &H8003 set up loop
1010 PRINT PEEK(I)  display contents
1020 NEXT I  continue
```

DESCRIPTION

The prefix "&H" is a special code that indicates "hexadecimal digits following". Hexadecimal notation is used in place of decimal or binary notation for Z-80 instruction codes, data relating to machine-language operation, and system addresses. The &H prefix may be followed by 1 to 4 hexadecimal digits. Each hexadecimal digit is 0 through 9 or A through F and represents a power of 16. The maximum hexadecimal value that can be defined in TRS-80 systems is &HFFFF, representing binary 1111111111111111, or decimal 65,535.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...&Odddddd...

EXAMPLES

```
1010 FOR I=&O100000 TO &O100003 setup
loop
1020 PRINT PEEK(I) print contents
1030 NEXT I loop
```

DESCRIPTION

The prefix "&O" is a special code that indicates "octal digits following". Octal notation is sometimes (rarely) used in place of decimal or binary notation for Z-80 instruction codes, data relating to machine-language operation, and system addresses. The &O prefix may be followed by 1 to 6 octal digits. Each octal digit is 0 through 7 and represents a power of 8. The maximum octal value that can be defined in TRS-80 systems is &O177777, equivalent to binary 1111111111111111, or decimal 65535. The prefix "&" is equivalent to "&O" and may be used in its place.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# ' remark text
line# ...' remark text...

EXAMPLES

```
1000 'THIS IS A REMARK LINE
1010 A=B 'AND SO IS THIS PORTION
```

DESCRIPTION

The single quote replaces the colon (:), REM commands. In effect, it is a shorthand way of creating a new REM statement, either at the beginning of a line or in the middle of a line. Using the single quote creates "pretty" listings that may be much more readable. The single quote may be placed anywhere in the line.

RELATED COMMANDS

REM

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# ...(...)...

EXAMPLES

```
1000 A=B/(C+D)
```

DESCRIPTION

Parentheses are used to denote the order of operations in expressions. In the example above, the result should be B/(C+D); if the parentheses were not included the operation would become B/C, followed by the addition of D. BASIC always evaluates the expressions inside parentheses before evaluating the rest of the expression. Parentheses may be "nested", that is, there may be many levels of parentheses, one within another. BASIC always works from the innermost parentheses out in evaluating parentheses.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...**. . .

EXAMPLES

```
1000 C=3.14159*D   find circumference
1010 C=SQR(A*A+B*B) find length of
hypotenuse
```

DESCRIPTION

The special character "*" is reserved as a BASIC operator signifying multiplication, except for the Model I/III Level I, where it is also a logical "AND" operator. It should not be used in variable names or in any other context other than within text strings enclosed by quotes. "*" may be used any number of times within a BASIC statement as long as it is not immediately followed by another operator.

RELATED COMMANDS

*(AND)

SYSTEM

I, LVL I	•
I, LVL II	
I, Disk	
II	
III, LVL I	•
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...(expression) * (expression)...*

EXAMPLES

```
1000 IF (A<2) * (B>5) THEN PRINT
''HELP!''
1010 IF A * 3=3 THEN GOTO 8000
```

DESCRIPTION

In the Model I/III Level I, "*" is an abbreviation for the AND function in addition to representing a multiplication operator. AND is used as a relational operator and for bit manipulation. In the first use, AND compares two constants, variables, or expressions. If both expressions are true, then the AND function is true. In the example above, (A<2) * (B>5) is true only if variable A is less than 2 AND variable B is greater than 5. The THEN action would only be taken if both expressions were true (expression 1 AND expression 2). In the bit manipulation case, AND is used to logically AND integer variable bits, considered to be binary numbers. An AND of binary values produces a 1 for each bit position only if both operands have a 1 bit in that bit position. An AND of the two binary values 10100000 and 11001111 would produce a result of 10000000. The AND in this application can be used to test bits, mask out fields, and perform other bit-wise operations.

RELATED COMMANDS

*. +(OR)

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...expression+expression...

EXAMPLES

```
1000 C=1.5+32+N+M  find total
```

DESCRIPTION

The special character "+" is reserved as the sign of a constant or a BASIC operator signifying addition or string concatenation. (It is also used in the Model I/III Level I to specify a logical "OR".) It should not be used in variable names or in any other context other than within text strings enclosed by quotes. "+" may be used any number of times within a BASIC statement as long as it is not immediately followed by another operator. When used as an arithmetic operator, it has the same effect as the usual "plus" sign - it adds two quantities, which may be any mixture of constants, variables, or expressions. When used as a string concatenation operator (not a Model I/III Level I function), it joins two strings. The result string is made up of the first string appended by the second string. If A$="NOW IS THE TIME" and B$="FOR ALL GOOD PROGRAMMERS...", then C$=A$+B$ would set C$ equal to "NOW IS THE TIME FOR ALL GOOD PROGRAMMERS..." When used as a sign, it must be immediately followed by numeric data.

RELATED COMMANDS

+(OR)

SYSTEM

I, LVL I	•
I, LVL II	
I, Disk	
II	
III, LVL I	•
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...(expression) + (expression)...

EXAMPLES

```
1000 IF (A<2) + (B>5) THEN PRINT
"HELP!"
1010 A=A + 8 set bit 3
```

DESCRIPTION

In the Model I/III Level I, "+" is an abbreviation for the OR function along with representing an addition operator. OR is used as a relational operator and for bit manipulation. In the first use, OR compares two constants, variables, or expressions. If either expression is true, then the OR function is true. In the example above, (A<2) + (B>5) is true if variable A is less than 2 OR variable B is greater than 5. The THEN action would only be taken if either expressions was true (expression 1 + expression 2). In the bit manipulation case, OR is used to logically OR integer variable bits, considered to be binary numbers. An OR of binary values produces a 1 for each bit position if either operand has a 1 bit in that bit position. An OR of the two binary values 10100000 and 11001111 would produce a result of 11101111. The OR in this application can be used to test bits, set individual bits, and perform other bit-wise operations.

RELATED COMMANDS

*(AND), +

SYSTEM

I, LVL I	●
I, LVL II	●
I, Disk	●
II	●
III, LVL I	●
III, LVL III	●
III, Disk	●
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

*line#...*PRINT *item1,item2,...*
*line#...*LPRINT *item1,item2,...*

EXAMPLES

```
1000 PRINT A,
1010 PRINT "NUMBER IS ";N,"NEXT
IS ";M
```

DESCRIPTION

In addition to separating items in DATA lists and acting as a delimiter in certain BASIC commands, the comma has a special use in PRINT statements. It is used in PRINT and LPRINT statements to mean "tab to the next print zone". Both the video display and line printer lines are divided into "print zones", which are similar to predefined typewriter tabs. When a comma is encountered after a PRINT item, the BASIC interpreter will tab to the start of the next print zone. This allows for easy columnization of displayed and printed data items. The print zones are predefined and dependent upon the system used.

RELATED COMMANDS

;

SYSTEM

I, LVL I	●
I, LVL II	●
I, Disk	●
II	●
III, LVL I	●
III, LVL III	●
III, Disk	●
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

line#...expression - expression...

EXAMPLES

1000 L=L-1-N *find adjusted length*

DESCRIPTION

The special character "-" is reserved as a BASIC operator signifying subtraction or for negating values. It should not be used in variable names or in any other context other than within text strings enclosed by quotes. When used as an arithmetic operator, "-" may be used any number of times within a BASIC statement as long as it is not immediately followed by another operator. Its meaning is identical to the normal use of the subtract sign. When used to negate quantities, it must be immediately followed by a numerical constant, as in

```
1000 DATA -5,-67.89,+45,+1
```

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

used in Edit mode

EXAMPLES

```
EDIT.
```

DESCRIPTION

The period is used in Edit mode to mean "the current line". The command EDIT. will result in an Edit of the current line number. If line 400 was LISTed just prior to the EDIT., for example, EDIT. will invoke an edit of line 400.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...expression/expression...

EXAMPLES

```
1000 R=2*D/3.14159  find radians
1010 TO=SUM/100  find average score
```

DESCRIPTION

The special character "/" is reserved as a BASIC operator signifying division. It should not be used in variable names or in any other context other than within text strings enclosed by quotes. "/" may be used any number of times within a BASIC statement as long as it is not immediately followed by another operator.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# ...:...:...

EXAMPLES

```
1000 A=C*2 : B=C*64 : C$=A$ 'A
MULTIPLE-STATEMENT LINE
```

DESCRIPTION

The colon is used to create multiple-statement lines. A multiple-statement line, just as the name implies, has two or more separate statement groupings, with a common line number, as in the above example. All statements in the line will be executed in sequence, just as if they were separate lines. GOTOs or GOSUBs to the middle of the line, however, are not possible. When statements are appended to IF...THEN or IF...THEN...ELSE statements, the appended statements will not be executed unless the THEN or ELSE condition is satisfied. 1000 IF A=1 THEN B=0 : C=2 and 1010 IF A<>1 THEN B=1 ELSE B=0 : C=2 will set C equal to 2 only if A=1 (both cases).

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...PRINT *item1;item2;...*
line#...LPRINT *item1;item2;...*

EXAMPLES

```
1000 PRINT A;
1010 PRINT ""NUMBER IS "";N,""NEXT
IS "";M
```

DESCRIPTION

In addition to acting as a delimiter in certain BASIC commands, the semicolon has a special use in PRINT statements. It is used in PRINT and LPRINT statements to mean "do not space". Both the video display and line printer lines are divided into "print zones", which are similar to typewriter tabs. When a comma is encountered after a PRINT item, the BASIC interpreter will tab to the start of the next print zone. Using a semicolon, however, inhibits this tabbing and positions the video display cursor or the line printer print head over the next character position. This allows data items to be displayed or printed directly after related text or data items as in "PRINT ""NUMBER IS "";N," which would print

```
NUMBER IS  123.56
```

RELATED COMMANDS

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...expression<expression...

EXAMPLES

```
1000 IF (M-2)<N THEN GOTO 2000
1010 IF ZZ<23 THEN ZZ=ZZ+5 ELSE
ZZ=ZZ-1
1020 IF LEFT$(A$,1)<''M'' THEN
PRINT ''FIRST HALF''
```

DESCRIPTION

The < character is used either as a relational operator or as a string operator in BASIC. A relational operator compares two arithmetic quantities. When used as a relational operator, "<" stands for "less than" and is used to test one quantity against another, as in "IF A<23". In this use, < is used in the IF...THEN or IF...THEN...ELSE commands. When used as a string operator, < is used to test two strings against each other. Strings are compared on a character by character basis, with each character representing a "weight" determined by its ASCII value. ASCII values roughly follow alphabetic sequence. An "A" is "less than" a "B" in this context. The < is again used in the IF...THEN and IF...THEN...ELSE commands for string comparisons as in "IF A$<''CALIF''", which tests string A$ for "less than" string "CALIF".

RELATED COMMANDS

<=,<>,=,>,>=

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...expression<=expression...

EXAMPLES

```
1000 IF (M-2)<=N THEN GOTO 2000
1010 IF ZZ<=23 THEN ZZ=ZZ+5 ELSE
ZZ=ZZ-1
1020 IF LEFT$(A$,1)<=''M'' THEN
PRINT ''FIRST HALF''
```

DESCRIPTION

The <= characters are used either as a relational operator or as a string operator in BASIC. A relational operator compares two arithmetic quantities. When used as a relational operator "<=" stands for "less than or equal to" and is used to test one quantity against another, as in "IF A<=23". In this use, <= is used in the IF...THEN or IF...THEN...ELSE commands. When used as a string operator, <= is used to test two strings against each other. Strings are compared on a character by character basis, with each character representing a "weight" determined by its ASCII value. ASCII values roughly follow alphabetic sequence. An "A" is "less than" a "B" in this context. The <= is again used in the IF...THEN and IF...THEN...ELSE commands for string comparisons as in "IF A$<=''CALIF''", which tests string A$ for "less than or equal to" string "CALIF".

RELATED COMMANDS

<,<>,=,>,>=

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...expression<>expression...

EXAMPLES

```
1000 IF (M-2)<>N THEN GOTO 2000
1010 IF ZZ<>23 THEN ZZ=ZZ+5 ELSE
ZZ=ZZ-1
1020 IF LEFT$(A$,1)<>''M'' THEN
PRINT ''NOT M''
```

DESCRIPTION

The <> characters are used either as a relational operator or as a string operator in BASIC. A relational operator compares two arithmetic quantities. When used as a relational operator "<>" stands for "not equal to" and is used to test one quantity against another, as in "IF A<>23". In this use, <> is used in the IF...THEN or IF...THEN...ELSE commands. When used as a string operator, <> is used to test two strings against each other. Strings are compared on a character by character basis, with each character representing a "weight" determined by its ASCII value. ASCII values roughly follow alphabetic sequence. An "A" is "less than" a "B" in this context. The <> is again used in the IF...THEN and IF...THEN...ELSE commands for string comparisons as in "IF A$<>''CALIF''", which tests string A$ for "not equal to" string "CALIF".

RELATED COMMANDS

<,<=,=,>,>=

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# variable=expression

line#...expression=expression...

line#...string=string...

EXAMPLES

```
1000 PI=3.14159
1010 IF N=(23-M) THEN N=0
1020 IF A$=B$ THEN PRINT
''FOUND''
```

DESCRIPTION

The equals sign "=" is used to equate a variable to a quantity, as a relational operator, or as a string operator. When used as to equate a variable to a quantity, it separates the variable from a constant, a second variable, or an expression, and sets the variable on the left-hand side to the value of the argument on the right-hand side. When used as an arithmetic relational operator, it compares one expression with another, as in "IF (X-2)=1024". It is used in this context with the IF...THEN and IF...THEN...ELSE commands. When used as a string operator, it compares two strings with one another, as in "IF A$=B$+C$" or "IF A$=''FALSE''". It is also used in the IF...THEN or IF...THEN...ELSE commands as a string operator.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...expression>expression...

EXAMPLES

```
1000 IF X>101 THEN GOTO 1050
1010 IF ZZ>23 THEN ZZ=ZZ+5 ELSE
ZZ=ZZ-1
1020 IF LEFT$(A$,1)>''CA'' THEN
STOP
```

DESCRIPTION

The > character is used either as a relational operator or as a string operator in BASIC. A relational operator compares two arithmetic quantities. When used as a relational operator ">" stands for "greater than" and is used to test one quantity against another, as in "IF A>23". In this use, > is used in the IF...THEN or IF...THEN...ELSE commands. When used as a string operator, > is used to test two strings against each other. Strings are compared on a character by character basis, with each character representing a "weight" determined by its ASCII value. ASCII values roughly follow alphabetic sequence. A "Z" is "greater than" a "W" in this context. The > is again used in the IF...THEN and IF...THEN...ELSE commands for string comparisons as in "IF A$>''CALIF''", which tests string A$ for "greater than" string "CALIF".

RELATED COMMANDS

<,<=,<>,=,>=

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...expression>=expression...

EXAMPLES

```
1000 IF X>=101 THEN GOTO 1050
1010 IF ZZ>=23 THEN ZZ=ZZ+5 ELSE
ZZ=ZZ-1
1020 IF LEFT$(A$,1)>=''CA'' THEN
STOP
```

DESCRIPTION

The >= characters are used either as a relational operator or as a string operator in BASIC. A relational operator compares two arithmetic quantities. When used as a relational operator ">=" stands for "greater than or equal to" and is used to test one quantity against another, as in "IF A>=23". In this use, >= is used in the IF...THEN or IF...THEN...ELSE commands. When used as a string operator, >= is used to test two strings against each other. Strings are compared on a character by character basis, with each character representing a "weight" determined by its ASCII value. ASCII values roughly follow alphabetic sequence. A "Z" is "greater than" a "W" in this context. The >= is again used in the IF...THEN and IF...THEN...ELSE commands for string comparisons as in "IF A$>=''CALIF''", which tests string A$ for "greater than or equal to" string "CALIF".

RELATED COMMANDS

<,<=,<>,=,>

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*ABS*(expression)...*

EXAMPLES

```
1000 REM FIND X DISTANCE
1010 XD=ABS(X1-X2)
```

DESCRIPTION

ABS returns the absolute value of a constant, variable, or expression. It is a function that may be used anywhere within a BASIC statement.
ABS(X)=X for X equal to or greater than 0.
ABS(X)=-X for X less than 0. In other words, the result of the ABS is always positive.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...(expression) AND *(expression)...*

EXAMPLES

```
1000 IF (A<2) AND (B>5) THEN PRINT
"HELP!"
1010 IF (A AND 3=3) THEN GOTO 8000
```

DESCRIPTION

AND is used as a relational operator and for bit manipulation. In the first use, AND compares two constants, variables, or expressions. If both expressions are true, then the AND function is true. In the example above, (A<2) AND (B>5) is true only if variable A is less than 2 AND variable B is greater than 5. The THEN action would only be taken if both expressions were true (expression 1 AND expression 2). In the bit manipulation case, AND is used to logically AND integer variable bits, considered to be binary numbers. An AND of binary values produces a 1 for each bit position only if both operands have a 1 bit in that bit position. An AND of the two binary values 10100000 and 11001111 would produce a result of 10000000. The AND in this application can be used to test bits, mask out fields, and perform other bit-wise operations.

RELATED COMMANDS

NOT, OR

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...`ASC`*(string)...*

EXAMPLES

`1000 A=ASC(A$)` *get first character of A$ in numeric*

`1010 B=ASC(''NOW IS THE TIME'')` *get "N" in numeric*

DESCRIPTION

`ASC` finds the ASCII code of the first letter of the specified string. In other words it takes the string argument, strips off the first character, and returns it as a numeric value, rather than a string character. It is a partial "convert to numeric" as in `VAL`. In the second example above, `ASC` would take the string "NOW IS THE TIME", strip off the "N", and return the "N" as a decimal 78, the ASCII code for "N". `ASC` can be used for alphabetizing and other string processing. `ASC` performs the inverse of the `CHR$` function.

RELATED COMMANDS

`CHR$, STR$, VAL`

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...`ATN`*(expression)...*

EXAMPLES

`1000 PRINT ATN(X)*57.29578` *print angle*

DESCRIPTION

`ATN` finds the arctangent of the argument. The arctangent is the angle in radians of the argument, assumed to be a tangent value. The expression may be a constant, variable, or expression. The result of `ATN` is in radians. To find the result in degrees, multiply by 180/pi, or 57.29578. `ATN` is the inverse of the `TAN` function, which finds the tangent of an angle in radians.

RELATED COMMANDS

`TAN`

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

AUDIO ON
line# AUDIO ON
AUDIO OFF
line# AUDIO OFF

EXAMPLES

1000 AUDIO ON *turn on TV speaker*
3000 AUDIO OFF *turn off TV speaker*

DESCRIPTION

AUDIO ON routes the cassette output to the TV speaker. The TV speaker can now be used to monitor CLOADs and CLOADMs of cassette files. This can be helpful in positioning the tape and verifying that cassette data is valid. AUDIO OFF turns off the audio routing.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

AUTO
AUTO *line#*
AUTO *line#,increment*

EXAMPLES

AUTO 100,2 *number lines* 100,102,104,*etc.*

DESCRIPTION

AUTO invokes the automatic line numbering mode of BASIC. The BASIC interpreter will automatically display a line number, starting with the line# start specified in the AUTO command, and will increment the line numbers by the increment number specified in AUTO. AUTO is used primarily in creating new programs; the user fills out the remainder of the BASIC line, terminates it with ENTER, and then continues with the next AUTO line number. The line# and increment are optional. If the increment is not specified, the default increment is 10. If neither the line number nor increment are specified, the starting line number is 10. AUTO is not related to TRSDOS AUTO.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

BACKUP 0
BACKUP *source drive* TO *destination drive*

EXAMPLES

```
BACKUP 0
BACKUP 0 TO 1
```

DESCRIPTION

BACKUP is a Color Computer Disk BASIC command that duplicates the contents of one diskette on a second diskette. The backup is an exact copy of the original disk. If a single drive system is used, the "BACKUP 0" form of the command is used; the Backup program will prompt you to switch diskettes at the proper times. If you have two or more disk drives, either the BACKUP 0 or two-drive version of the command may be used. The backup is made from the diskette in the "source drive" to the diskette in the "destination drive".

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...CDBL*(expression)*...

EXAMPLES

1000 PRINT CDBL(I%/J%) *print 17 digits*

DESCRIPTION

CDBL forces processing in double precision, even though some of the variables involved may be integer or single-precision operands. CDBL is used whenever the result is required to be of double-precision accuracy (17 decimal digits of significance). Of course, if the processing done up to a particular point has been extensive, and only in single precision, CDBL cannot retrieve the lost digits of significance! In the example above CDBL (I%/J%) is accurate because both I% and J% are integer variables and have lost no significance in processing. Performing a CDBL(A/B) will in many cases be accurate only to single-precision accuracy as A and B are single-precision variables.

RELATED COMMANDS

CINT, CSNG

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*CHR$*(expression)...*

EXAMPLES

```
1000 PRINT ''ESCAPE
SEQUENCE'';CHR$(27);CHR$(101)
```

DESCRIPTION

The CHR$ function converts one numeric value to a one-character string. The one-character string can then be appended to other strings or used as a single-character string. CHR$ allows a way of specifying non-ASCII characters from the keyboard. Certain line printers expect to see numeric codes which have no keyboard equivalent; CHR$ permits embedding these codes in a string sent to the line printer. CHR$ can also be used to construct strings used for graphics purposes. CHR$ performs the inverse of the ASC function.

RELATED COMMANDS

ASC, STR$, VAL

SYSTEM

I, LVL I	
I, LVL I'	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*CINT*(expression)...*

EXAMPLES

`1000 A=CINT(B#)*CINT(C#)` *convert and multiply*

DESCRIPTION

CINT forces processing to be done in integer mode. The constant, variable, or expression is converted to an integer by the CINT function. Integer values are held in two bytes and may range from -32768 to +32767. The CINT converts the argument to an integer variable by using only the integer portion of the argument. If the argument were 3456.777, for example, the result of CINT would be 3456. CINT is used anytime that a variable or expression can be converted to integer to speed up processing.

RELATED COMMANDS

CDBL, CSNG

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# CIRCLE*(x,y),r circle*
line# CIRCLE*(x,y),r,c circle with color*
line# CIRCLE*(x,y),r,c,hw ellipse*
line# CIRCLE*(x,y),r,c,hw,start,end arc*

EXAMPLES

1000 CIRCLE(129,96),40 *radius 40 circle*
1010 CIRCLE(200,100),20,4,1,0,.25 *red arc*

DESCRIPTION

CIRCLE is used to draw a circle, ellipse, or arc at any point on the current graphics screen. The x and y parameters specify the center point for the circle, ellipse, or arc. The ranges of x and y are 0 through 255 and 0 through 191, respectively. The r parameter is the radius of the circle or 1/2 the width of the ellipse. The c parameter is the color code (1 through 8) for the figure. The hw parameter is the height/width ratio for the figure. A circle has hw=1, ellipses hw ratios from 0 through large values. The "start" and "end" parameters define the start and end points of the arc. Any value from 0 (three o'clock) through 1 (clockwise back to three o'clock) may be used to define the start and end points. Commas may be used in place of the c, hw, start, and end parameters. Defaults are c=foreground, hw=1, start=0, and end=1.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

CLEAR *N (Model I,II,III)*
CLEAR *N,M (Color Computer)*
line# CLEAR *N or* CLEAR *N,M*

EXAMPLES

1000 CLEAR 1000 *clear 1000 bytes for strings*
1010 CLEAR 100,16000 *clear 100 bytes for strings, protect memory*

DESCRIPTION

CLEAR clears all variables to 0 and sets aside a specified number of bytes of RAM for a "string storage area". This string storage area is used exclusively as a working storage area for string processing. Enough bytes should be set aside to handle the maximum number of characters in string variables during program execution. This is usually a trial and error computation. If too few characters are set aside, either an "out of string space" error will occur, or some time will be lost while the BASIC interpreter "cleans up" the string storage area to make room for new strings. In the Color Computer, a second parameter protects all RAM from a given address up to "top of RAM"; this area is normally used for storage of machine-language programs or buffers.

RELATED COMMANDS

FRE

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

CLOAD *"file name"*
CLOAD

EXAMPLES

```
CLOAD ""RATTAIL""
```

DESCRIPTION

CLOAD is used to load a BASIC program file from cassette. The file name, if used, must be in quotes. If no file name is specified in the CLOAD command, the next BASIC file from cassette will be loaded. If a file name is specified, the cassette tape will be searched for that specific file name. File names are one character long in the Model I and III and up to six characters long in the Color Computer. As BASIC searches for the proper file, it will display all files encountered on the video display. When the next or named file is found, it is assumed to be a BASIC file, and will replace any current BASIC program in RAM. In addition to initializing the BASIC program area, a CLOAD also resets all variables to 0 and initializes other BASIC program parameters. For systems with two cassettes, see CLOAD#-.

RELATED COMMANDS

CLOAD#-, CLOAD?, CSAVE

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

CLOAD#-1,*"file name"*
CLOAD#-2,*"file name"*

EXAMPLES

```
CLOAD#-1,""RATTAIL""
```

DESCRIPTION

CLOAD#- is used to load a BASIC program file from cassette when two cassettes are used in the system. The file name, if used, must be in quotes. If no file name is specified in the CLOAD#- command, the next BASIC file from cassette will be loaded. If a file name is specified, the cassette tape will be searched for that specific file name. File names are one character long in the Model I. When the next or named file is found, it is assumed to be a BASIC file, and will replace any current BASIC program in RAM. In addition to initializing the BASIC program area, a CLOAD#- also resets all variables to 0 and initializes other BASIC program parameters.

RELATED COMMANDS

CLOAD?#-, CSAVE#-

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
CLOAD? "file name"
CLOAD
```

EXAMPLES

```
CLOAD? ''RATTAIL''
```

DESCRIPTION

CLOAD? is used to compare a program on cassette with the BASIC program in RAM. It is normally used directly after a CSAVE operation to compare the BASIC file just saved with the contents of RAM. This ensures that the BASIC program will not be destroyed before a valid copy has been saved on cassette. The "file-name" is optional. If no file name is specified, then the next file on cassette will be compared with the BASIC program in RAM. If a file name is specified, the BASIC interpreter will search cassette until the specified file is found. If the file on tape is not identical with the contents of RAM, a "BAD" message will be displayed and another CSAVE operation must be done. The BASIC program in RAM is not altered during the comparison process. If the system used has two cassettes, see CLOAD?#.

RELATED COMMANDS

CLOAD, CLOAD?#-, CSAVE

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
CLOAD?#-1,"file name"
CLOAD?#-2,"file name"
```

EXAMPLES

```
CLOAD?#-2,''RATTAIL''
```

DESCRIPTION

CLOAD?#- is used to compare a program on cassette with the BASIC program in RAM for those systems that have more than one cassette. It is normally used directly after a CSAVE#- operation to compare the BASIC file just saved with the contents of RAM. This ensures that the BASIC program will not be destroyed before a valid copy has been saved on cassette. The "file-name" is optional. If no file name is specified, then the next file on cassette will be compared with the BASIC program in RAM. If a file name is specified, the BASIC interpreter will search cassette until the specified file is found. If the file on tape is not identical with the contents of RAM, a "BAD" message will be displayed and another CSAVE#- operation must be done. The BASIC program in RAM is not altered during the comparison process. The #-1 command will compare from cassette 1 and the CLOAD?#-2 command will compare from cassette 2.

RELATED COMMANDS

CLOAD#-, CSAVE#-

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

```
CLOADM
CLOADM"filename"
CLOADM "filename", offset
```

EXAMPLES

`CLOADM ''GRAPHC''` *load file "GRAPHC" into RAM*

DESCRIPTION

CLOADM is used to load a machine-language file from cassette tape. The cassette tape file may have been generated by the Color Computer Editor/Assembler or be in a format compatible with the CLOADM function. When CLOADM is used alone, the next file on cassette is assumed to be a machine-language file and is loaded into RAM. When the "CLOADM"filename" " format is used, the CLOADM routine will search for the specified file name on cassette. When it finds the file, it will be loaded into RAM as a machine-language file. When the "CLOADM"filename", offset" format is used, the named machine-language file will be loaded into RAM at the normal locations specified in the file plus the offset value. The offset value may be any value except those that cause the load address to be in "non-existent" RAM.

RELATED COMMANDS

EXEC

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# `CLOSE` *buf#1,buf#2,...,buf#n*

EXAMPLES

`1000 CLOSE 1.3` *close files for buffers 1 and 3*

DESCRIPTION

CLOSE "closes" a disk file or files. A disk file is normally first OPENed for reading or writing. The OPEN command causes BASIC to find the file name in the directory and to establish the disk location of the file, type of file, and other parameters. OPEN also allocates a RAM "buffer" to be used with the file. The RAM buffer is the memory area used for reading or writing disk sectors. Buffers are allocated by number, and the OPEN associates a specified file name with the buffer number. After the records of the file have been read or written, a CLOSE "flushes" any remaining data in a buffer for a write and properly terminates file operations for the designated buffer or buffers. The "buf#" parameters specify the buffer numbers, and hence, the files to be closed. One or more buffer numbers may be specified.

RELATED COMMANDS

OPEN

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# CLS
line# CLS c *Color Computer*

EXAMPLES

```
1000 CLS
```
clears video display

```
2000 CLS 3
```
clears display to blue (Color Computer)

DESCRIPTION

Model I/II/III: CLS clears the entire video display screen by outputting blanks to each of the screen character positions. Note that this is an ASCII 32, an alphabetic blank, rather than a graphics character. The screen cursor is then positioned in the upper left-hand corner of the screen.
Color Computer: CLS clears the entire screen to a specified color, c. The c parameter is a color code of 0 through 8 (black, green, yellow, blue, red, buff, cyan, magenta, orange).

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
CMD"A"
```

EXAMPLES

```
CMD"A"
```

DESCRIPTION

The CMD"A" command allows you to return to TRSDOS from BASIC. Typing in "CMD"A"" at any time when in the command mode of BASIC causes a return to TRSDOS.

```
>CMD"A"
OPERATION ABORTED
TRSDOS READY
```

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

```
CMD"B","ON"
CMD"B","OFF"
```

EXAMPLES

`CMD"B","ON"` *enables the BREAK key operation*

`CMD"B","OFF"` *disables the BREAK key operation*

DESCRIPTION

CMD"B" is used to enable or disable the BREAK key. The BREAK key is normally used to stop execution of a BASIC program. When the BREAK key is disabled with a CMD"B","OFF", the BREAK key will be ignored except during cassette, printer, or serial input/output. CMD"B" can be used to "lock out" the BREAK key to prevent erroneous stops of critical BASIC programs. The double quotes around ON and OFF are necessary. The BREAK key will be enabled upon a return to TRSDOS.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

```
CMD"C"
CMD"C",R
CMD"C",S
```

EXAMPLES

`CMD"C",S` *compress program by deleting spaces*

DESCRIPTION

CMD"C" is a command to "compress" a program by deleting remarks and/or spaces. BASIC program remarks take up about one byte in RAM for every REM character. They are most useful during program debugging and may be deleted after a final version of the program has been reached. Spaces help readability, but also take up one byte of RAM for every space. If the CMD"C" format is used, text from both REMs (and ' type remarks) and spaces are deleted from the BASIC. If the other formats are used either remarks or spaces are deleted. All spaces except those inside string literals will be deleted. String literals (such as A$="STRING LITERAL") must have double quotes at both beginning and end for the command to function properly.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk ●
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

CMD"D"

EXAMPLES

CMD"D" *load* DEBUG *from disk*

DESCRIPTION

CMD"D" loads the DEBUG program from disk. DEBUG may be entered by pressing the BREAK key at any time after DEBUG has been loaded. DEBUG is used to examine memory, execute machine-language programs, and perform other non-BASIC tasks. BASIC program text and variables will be lost after transfer of control to DEBUG.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk ●
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

CMD"D:*d*"

EXAMPLES

CMD"D:1" *display directory of drive 1*

DESCRIPTION

CMD"D" is a BASIC command similar to the TRSDOS DIR command. It allows the user to display a diskette directory from inside BASIC without transferring to TRSDOS. The "d" parameter is the drive number, 0 through 3. Only unprotected, visible files will be displayed.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

```
CMD""E""
```

EXAMPLES

`CMD""E""` *display last TRSDOS error*

DESCRIPTION

`CMD""E""` displays the last TRSDOS error from within `BASIC`. It is a way of getting further information about the type of TRSDOS error that occurred, rather than a "blanket" statement. If, for example, `BASIC` returned a "`DISK I/O ERROR`", entering `CMD""E""` would expand on this by displaying the last TRSDOS error message of "`DISK DRIVE NOT IN SYSTEM`". This message would not have been displayed during `BASIC` program execution.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

`CMD""I"",`*"command"*
line# `CMD""I"",`*"command"*

EXAMPLES

```
1000 A$=""DIR""
1010 CMD""I"",A$
```
exit to TRSDOS and do dir

DESCRIPTION

`CMD""I""` returns control to TRSDOS from `BASIC` and passes a command. The command is executed as the first TRSDOS action.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line# CMD"J",*"mm/dd/yy", string*
line# CMD"J",*"-yy/ddd", string*

EXAMPLES

1000 CMD"J",12/05/81,A$ *convert date*

DESCRIPTION

CMD"J" converts a given date to "day-of-the-year" format or converts the day of the year to mm/dd/yy format. The "dd" or "ddd" parameter is the day. The "mm" and "yy" parameters are month and year, respectively. This command is used to convert the mm/dd/yy format to ddd format or the yy/ddd format to mm/dd/yy format. The result of CMD"J" is the format opposite to the one specified after the CMD"J". The result is held in the specified string. CMD"J" is handy for converting to and from "Julian" format (yy,mmm) where the day of the year is 1 through 366. Julian format facilitates processing of elapsed time. The minus sign prior to the yy/ddd is required. The command CMD"J","12/05/81",A$ produces A$="339". The command CMD"J","-81/300",A$ produces A$="10/27/81".

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

CMD"L",*"filename"*
CMD"L",*string*

EXAMPLES

1000 CMD"L","ASSEMP:1" *load machine language*

DESCRIPTION

CMD"L" loads in a machine-language file created by the TRSDOS DUMP command or Disk Editor/Assembler. The machine-language file would normally contain code to be interfaced to BASIC through the DEFUSRn and USRn commands. The machine-language code cannot overlay the RAM area protected by the MEMORY SIZE? prompt. If the filename format is used, the filename must be enclosed by quotes; if the string format is used, quotes are not required. CMD"L",A$ will load in the file named in A$, assumed to be a machine-language file.

RELATED COMMANDS

DEFUSRn, USRn

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line# CMD"O",*integer variable,string array(start)*

EXAMPLES

```
1000 Z%=100
1010 CMD"O",Z%,A$(20)   sort array
```

DESCRIPTION

CMD"O" sorts a one-dimensional string array from a specified starting element number through a specified length. The sort will sequence the array entries so that they are ordered in "ascending sequence" based upon their ASCII codes and other values. Normal string array entries will contain ASCII representation of string variables. If the entries of the string array contain non-ASCII characters, such as control codes or graphics characters, the sort will be on the basis of their numerical values from 0 through 256. The "string array(start)" parameter defines the starting element of the string array. This may be the first element (0) or any element of the array. The integer variable parameter defines the number of elements from this start element. The sort will be performed on the array elements from the start through the start+n-1. The array element strings may be of mixed lengths.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line# CMD"P",*string*

EXAMPLES

```
1000 CMD"P",A$          get printer status
```

DESCRIPTION

CMD"P" reads in the system printer status. The printer status is returned as a string variable, the string parameter. This command is used to test the ready condition of the system line printer before using an LPRINT or other command. The line printer may not be ready because it is "off-line" or because of an error condition such as being out of paper. Printer status can be tested by converting the string result to numeric by the VAL command, and ANDing with 240 to obtain the most significant 4 bits of the status. Generally, if the result of the VAL conversion and ANDing is not 48 (binary 0011XXXX), the printer is not ready, although this depends upon the printer type in your system. Sample code is

```
1000 CMD"P",A$
1010 A=VAL(A$) AND 240
1020 IF A<>48 THEN PRINT "PRINTER
NOT READY"
```

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
CMD"R"
```

EXAMPLES

`CMD"R"` *turns on the real-time clock*

DESCRIPTION

`CMD"R"` is used to turn on the real-time clock from `BASIC`. The system real-time clock displays the 24-hour time at the upper right-hand corner of the screen. The time can be set by the TRSDOS `TIME` command. When the real-time clock is on, the time will be updated in fractions of a second and displayed in seconds. The real-time clock is always running except during cassette or disk input/output; `CMD"R"` simply enables the time display during all `BASIC` activity. The display can be disabled by the `CMD"T"` command.

RELATED COMMANDS

`CMD"T"`, `TIME (TRSDOS)`

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
CMD"S"
```

EXAMPLES

`CMD"S"` *return to TRSDOS*

DESCRIPTION

`CMD"S"` is used to return to TRSDOS from Disk `BASIC`. Executing `CMD"S"` will exit Disk `BASIC` and reload TRSDOS.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk •
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

```
CMD"T"
```

EXAMPLES

`CMD"T"` *disables the real-time clock display*

DESCRIPTION

`CMD"T"` turns off the system real-time clock from the command mode of `BASIC`. The real-time clock updates the time in fractions of a second and displays the 24-hour time in seconds in the upper right-hand corner of the screen. It is always running, except during cassette or disk input/output; using `CMD"T"` simply disables the screen display.

RELATED COMMANDS

`CMD"R"`, `TIME (TRSDOS)`

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line# `CMD"X"`,*reserved wd*
line# `CMD"X"`,*"string"*

EXAMPLES

`1000 CMD"X",PRINT` *find all* `PRINT`*s*

DESCRIPTION

`CMD"X"` will search the current `BASIC` program in RAM for either a reserved word such as `PRINT` or `GOTO`, or for a given string literal such as "EMPLOYEE #". The line numbers of all occurrences of the reserved word or string literal will then be listed on the display. `CMD"X"` can be used as a general search routine to facilitate changes in a `BASIC` program. A search for `PRINT`, for example, could easily be done and the `PRINT`s could then be changed to `LPRINT`s. The reserved word must not be in quotes; a string literal must be enclosed in quotes.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	●
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
CMD"Z","ON"
CMD"Z","OFF"
```

EXAMPLES

`CMD"Z","ON"` *turn on printer output*

DESCRIPTION

CMD"Z" is used to enable or disable simultaneous display and printer output. When CMD"Z","ON" is given, all output going to the display is also sent to the system line printer. The printer must be in a "ready" condition. Due to differences in character interpretation, display output sent to the line printer may cause unpredictable results, but in general, any text data sent to the screen will be properly printed on the system line printer. The printer output is disabled by CMD"Z","OFF". This command can be used to provide a hard copy of BASIC program output which normally would be displayed.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	●
CC, Disk	●

FORMAT

line# COLOR *foreground,background*

EXAMPLES

`1000 COLOR 2.3` *select yellow on blue*

DESCRIPTION

COLOR is used to select the foreground and background colors in either the text or graphics modes. The background is the field upon which figures can be drawn; the foreground is the color used to draw the figures. The color codes used are the standard Color Computer codes of 0 through 8 - black, green, yellow, blue, red, buff, cyan, magenta, and orange, respectively. The color codes used in the command must be valid colors in the current mode. The current mode depends upon the current SCREEN command in force (text or graphics) and the graphics mode (PMODE). The background may be selected to be the same color as the "border" color, in which case there will be no border around the screen.

RELATED COMMANDS

PMODE. SCREEN

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

CONT

EXAMPLES

CONT *(continue after stop)*

DESCRIPTION

CONT is an abbreviation for "continue". Continue is used after a STOP command has been executed. The STOP causes a temporary program halt, valuable for examination of variables or "breakpointing" during debugging. CONT is used after the STOP to continue the program from the point at which the STOP occurred. All variables will be intact when the CONT is executed. CONT is used in the command mode after a STOP has taken place.

RELATED COMMANDS

STOP

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

COPY *"filename1"* TO *"filename2"*

EXAMPLES

```
COPY ''TRANSFIL/BAS:0'' TO
''TRANSFIL/BAS:1''
```

DESCRIPTION

COPY is a Color Computer Disk BASIC command. It copies a complete file from one diskette to another diskette under the same or different file name, or copies a file to the same diskette under a different name. COPY is used to backup a single file, or to duplicate a file on the same or different diskettes. The file defined by "filename1" is copied as "filename2". Each filename must have an extension. The extension follows the main file name and is a three-character designator preceded by a slash character. The drive number is optional and is used only when the copy will be done between two different disk drives.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*COS*(expression)...*

EXAMPLES

1000 A=COS(X+3.14159/2) *sets variable A equal to cosine of X+pi/2 (in radians)*
2000 ND=COS(X*.01745329) *sets variable ND equal to cosine of X (in degrees)*

DESCRIPTION

COS finds the cosine of a given constant, variable, or expression. The quantity is assumed to be in radians (180/pi degrees). COS is a "function" and may be used anywhere within a BASIC statement as long as the argument is enclosed within parentheses. Multiply by .01745329 to convert degrees to radians. Standard trigonometric rules apply in regard to the sign of the result.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

CSAVE *"file name"*
CSAVE

EXAMPLES

CSAVE ''RATTAIL''

DESCRIPTION

The CSAVE command is used to save the current BASIC program in RAM on cassette tape. The tape must be positioned beyond the leader. Note the position of the tape by the tape counter for restart. If a "file name" is specified, the contents of RAM will be written out as a file called "file name". If no file name is specified, the name "NONAME" will be used. Legitimate file names for the Model I/III are single character names. Legitimate names for the Color Computer are 1 to 6 character names. CLOAD? may be used to verify that the file was written properly. A subsequent CLOAD will reload the BASIC program and "overlay" any current BASIC program in RAM. See CSAVE#- for systems with more than one cassette.

RELATED COMMANDS

CLOAD?, CSAVE#-

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
CSAVE#-1,"file name"
CSAVE#-2,"file name"
```

EXAMPLES

```
CSAVE#-,''RATTAIL''
```

DESCRIPTION

The CSAVE#- command is used to save the current BASIC program in RAM on cassette tape on those systems that have more than one cassette. The tape must be positioned beyond the leader. Note the position of the tape by the tape counter for restart. If a "file name" is specified, the contents of RAM will be written out as a file called "file name". If no file name is specified, the name "NONAME" will be used. Legitimate file names for the Model I are single character names. CLOAD?#- may be used to verify that the file was written properly. A subsequent CLOAD#- will reload the BASIC program and "overlay" any current BASIC program in RAM.

RELATED COMMANDS

CLOAD#-, CLOAD?#-

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

CSAVEM *"filename",startaddr,endaddr,execaddr*

EXAMPLES

```
CSAVEM ''SORTPR'',&H3000,&H3FFF,
&H3000
```

DESCRIPTION

CSAVEM is used to save a machine-language program in RAM as a cassette file. The "filename" parameter is a standard cassette file name. CSAVEM can be used to save any binary data in RAM whether it is a 6809E machine-language program, data, or both. The startaddr parameter specfies the starting address of the data to be saved. The endaddr parameter specifies the end of the data. The execaddr specifies the address of the start of the program, if applicable, or to a dummy parameter. The resulting file is stored as a binary file and can be loaded and executed by the CLOADM and EXEC commands.

RELATED COMMANDS

CLOADM,EXEC

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*CSNG*(expression)...*

EXAMPLES

1000 PRINT CSNG(ST#*NM#) *convert to sp and print*

DESCRIPTION

CSNG converts a constant, variable, or expression to single precision. Single-precision numbers can hold up to 7 decimal digits and occupy four bytes of storage. CSNG is used whenever it is convenient to convert from integer precision or double precision to single precision.

RELATED COMMANDS

CDBL, CINT

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*CVD*(string)...*

EXAMPLES

1000 A#=CVD(BAL$) *convert* BAL$ *to numeric*

DESCRIPTION

CVD is used to convert a string variable to a double-precision variable. CVD is normally used to retrieve a data value from a random-file buffer. The typical sequence in retrieving data from a random-file buffer is to define the fields in a random-access buffer with FIELD, to read in the disk file (see GET), and then to retrieve data with CVD, CVI, or CVS. CVD is the inverse of MKD$, which is normally used to store double-precision data in the random-file buffer in character string form. The CVD function converts a field from the buffer to numeric form. The field is assumed to contain an 8-character string created by MKD$. An error or invalid results would normally occur for a field size other than 8 characters. CVD can also operate on a string variable other than a FIELD variable. In this case the variable should have been created by MKD$.

RELATED COMMANDS

FIELD, MKD$

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*CVI*(string)...*

EXAMPLES

`1000 A%=CVI(EMP$)` *convert* EMP$ *to numeric*

DESCRIPTION

CVI is used to convert a string variable to an integer variable. CVI is normally used to retrieve a data value from a random-file buffer. The typical sequence in retrieving data from a random-file buffer is to define the fields in a random-access buffer with FIELD, to read in the disk file (see GET), and then to retrieve data with CVD. CVI, or CVI. CVI is the inverse of MKI$, which is normally used to store integer data in the random-file buffer in character string form. The CVI function converts a field from the buffer to numeric form. The field is assumed to contain a 2-character string created by MKI$. An error or invalid results would normally occur for a field size other than 2 characters. CVI can also operate on a string variable other than a FIELD variable. In this case the variable should have been created by MKI$.

RELATED COMMANDS

FIELD, MKI$

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

*line#...*CVN*(string)...*

EXAMPLES

`1000 A=CVN(ZIP$)` *convert* ZIP$ *to numeric*

DESCRIPTION

CVN is used to convert a string variable to a numeric variable. CVN is normally used to retrieve a data value from a direct-file buffer. The typical sequence in retrieving data from a direct-file buffer is to define the fields in direct-file buffer with FIELD, to read in the disk file (see GET), and then to retrieve data with CVN. CVN is the inverse of MKN$, which is normally used to store numeric data in the direct-file buffer in character string form. The CVN function converts a field from the buffer to numeric form. The field is assumed to contain a 5-character string created by MKN$. An error or invalid results would normally occur for a field size other than 5 characters. CVN can also operate on a string variable other than a FIELD variable. In this case the variable should have been created by MKN$.

RELATED COMMANDS

FIELD, MKN$

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...CVS(string)...

EXAMPLES

`1000 A=CVS(ZIP$)` *convert* `ZIP$` *to numeric*

DESCRIPTION

CVS is used to convert a string variable to a double-precision variable. CVS is normally used to retrieve a data value from a random-file buffer. The typical sequence in retrieving data from a random-file buffer is to define the fields in a random-access buffer with FIELD, to read in the disk file (see GET), and then to retrieve data with CVD, CVI, or CVI. CVS is the inverse of MKS$, which is normally used to store single-precision data in the random-file buffer in character string form. The CVS function converts a field from the buffer to numeric form. The field is assumed to contain a 4-character string created by MKS$. An error or invalid results would normally occur for a field size other than 4 characters. CVS can also operate on a string variable other than a FIELD variable. In this case the variable should have been created by MKS$.

RELATED COMMANDS

FIELD, MKS$

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...x.xxxxDyy...

EXAMPLES

```
1000 A#=3.1415265358979323D+30
1010 ZZ#=1.76D-5
```

DESCRIPTION

D is used to denote double-precision numbers with scientific notation. The format of such a number consists of a fraction or mixed number, a "D", and a power of ten. The power of 10 may be positive (plus sign or no leading sign) or negative (negative sign). The fraction or mixed number may consist of up to 17 decimal digits. The decimal point may be located anywhere within the number. The decimal point is optional. The variable associated with the double-precision number must have a "#" type suffix, or be defined in a DEFDBL range (i.e. it must be a double-precision variable).

RELATED COMMANDS

#, DEFDBL

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# DATA *item 1, item 2, item 3, item 4, ... item N*

EXAMPLES

1000 DATA 5.2, 2, -3, 5, -1 *defines a list of 6 numeric items*

2000 DATA ORANGE,PEACH,PEAR *defines a list of three string items*

3000 DATA 5,PLUM,-2,7.58,6,PEAR, -5,-10.2 *defines a mixed list*

DESCRIPTION

DATA is used to define a list of numeric or string values to be used in the program. More than one DATA statement results in one large list. Values can be read by using the READ command. RESTORE is used to "reset the pointer" to the beginning of the list. The following statements read 1, -2.5, and PEAR into variables A, B, and A$:

1000 DATA 1,-2.5,PEAR *establishes list*
1010 READ A,B,A$ *reads values*
1020 RESTORE *resets pointer*

Double quotes must enclose a string value if the string has leading blanks, commas, or colons.

RELATED COMMANDS

READ, RESTORE

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...DATE$...

EXAMPLES

1000 PRINT ''TODAY'S DATE IS''; DATE$

DESCRIPTION

DATE$ returns the current date and information about the date as a text string. When TRSDOS is started up, the operator enters the current date. DATE$ returns this information in BASIC. The format of the DATE$ string is WWWMMMDDYYYYJJJXXY where WWW is the day of the week, MMM is the month, DD is the numbered day of the month, YYYY is the year, JJJ is the Julian day (numbered day of the year), XX is the numbered month of the year, and Y is the numbered day of the week. A typical string returned by DATE$ is: WedDec301981364122. Weeks start with Monday, the 0th day; all other parameters count from 1.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	●
II	●
III, LVL I	
III, LVL III	
III, Disk	●
CC, BASIC	
CC, Ext BASIC	●
CC, Disk	●

FORMAT

line# DEF FN*name(arg1,arg2,...,argn)=formula*

EXAMPLES

```
1000 DEF FNZ(A,B)=SQR(A*A+B*B)
```

DESCRIPTION

DEF FN is used to define a function. A function is a predefined operation that can be "invoked" by using the characters "FN" followed by the function name. Functions are useful if the same basic operation is repeated many times within a BASIC program. In the above example, suppose that the operation SQR(A*A+B*B) were to be repeated at 100 different places in a BASIC program. Defining it as DEF FNZ would permit code such as "2000 PRINT FNZ(101,50)"; the "FNZ" would execute the function called "Z" and perform SQR(101*101+50*50). The name parameter may be any variable name; any variable type suffix may be used, such as A%, A!, or A$. The arg parameters define the arguments to be used in the function; they are "dummies" in the DEF FN command and serve only as "place markers" for definition of the procedure. The dummies do not affect variable values. Only one argument may be used in the Color Computer.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	●
I, Disk	●
II	●
III, LVL I	
III, LVL III	●
III, Disk	●
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# DEFDBL *letter range*

EXAMPLES

```
1000 DEFDBL A-B
3000 DEFDBL I-K
```

DESCRIPTION

DEFDBL defines all variables within the specified letter range as double-precision numeric variables (17 decimal digits of precision stored, 16 displayed). Variables with type suffixes of "%", "!", "$", or "E", however, are not affected. The letter range defines a range of letters for the beginning letter of the variable. A letter range of I-K, for example, would include I, J, and K. After definition of this letter range by a DEFDBL, all variables beginning with I, J, or K would automatically be assumed to be double-precision variables, except for those with type suffixes. DEFDBL is a convenient way to define a range of variables as double-precision variables without having to define each variable separately with the # type suffix. DEFDBL would normally be used at the beginning of a BASIC program.

RELATED COMMANDS

!,#,$,%,DEFINT, DEFSNG, DEFSTR, E

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# DEFINT *letter range*

EXAMPLES

```
1000 DEFINT A-B
3000 DEFINT I-K
```

DESCRIPTION

DEFINT defines all variables within the specified letter range as integer variables (capable of holding -32768 to +32767). Variables with type suffixes of "#", "!", "D", "$", or "E", however, are not affected. The letter range defines a range of letters for the beginning letter of the variable. A letter range of I-K, for example, would include I, J, and K. After definition of this letter range by a DEFINT, all variables beginning with I, J, or K would automatically be assumed to be integer variables, except for those with type suffixes. DEFINT is a convenient way to define a range of variables as integer variables without having to define each variable separately with the % type suffix. DEFINT would normally be used at the beginning of a BASIC program.

RELATED COMMANDS

!,#,$,%,D, DEFDBL, DEFSNG, DEFSTR, E

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# DEFSNG *letter range*

EXAMPLES

```
1000 DEFSNG A-B
3000 DEFSNG I-K
```

DESCRIPTION

DEFSNG defines all variables within the specified letter range as single-precision variables (7 decimal digits of precision stored, 6 displayed). Variables with type suffixes of "%", "#", "D", or "$", however, are not affected. The letter range defines a range of letters for the beginning letter of the variable. A letter range of I-K, for example, would include I, J, and K. After definition of this letter range by a DEFSNG, all variables beginning with I, J, or K would automatically be assumed to be single-precision variables, except for those with type suffixes. Single-precision variables are the "default" mode for BASIC variables, and DEFSNG would not have to be used except to redefine variables that were previously assigned to other variable types.

RELATED COMMANDS

!, #, $, %, D, DEFDBL, DEFINT, DEFSTR, E

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# `DEFSTR` *letter range*

EXAMPLES

```
1000 DEFSTR A-B
3000 DEFSTR I-K
```

DESCRIPTION

DEFSTR defines all variables within the specified letter range as string variables. Variables with type suffixes of "%", "!", "#", "D", or "E", however, are not affected. The letter range defines a range of letters for the beginning letter of the variable. A letter range of I-K, for example, would include I, J, and K. After definition of this letter range by a DEFSTR, all variables beginning with I, J, or K would automatically be assumed to be string variables, except for those with type suffixes. DEFSTR is a convenient way to define a range of variables as string variables without having to define each variable separately with the $ type suffix. DEFSTR would normally be used at the beginning of a BASIC program.

RELATED COMMANDS

!, #, $, %, D, DEFINT, DEFDBL, DEFSNG, E

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# `DEFUSR`*n*`=address`

EXAMPLES

`1000 DEFUSR3=&H8000` *define subroutine for Model I*

DESCRIPTION

DEFUSR is used to define the location of a machine-language subroutine. The subroutine consists of machine language for the system in use. The n parameter in the DEFUSR command may be any number from 0 through 9; this allows up to 10 machine-language subroutines to be defined for interface to BASIC programs. The address value on the right-hand side of the DEFUSR command is the starting point for the machine-language code. The machine-language subroutine may consist of any number of instructions. The subroutine is called by the USRn call, in which n matches the n of the DEFUSR. USR3, for example, would match the DEFUSR3 definition.

RELATED COMMANDS

USRn

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC •
CC, Disk •

FORMAT

`DEL -`
`DEL` *line#–line#*
`DEL` *line#–*
`DEL -`*line#*
line# `DEL` *line#–line#*

EXAMPLES

`DEL 100-` *delete lines* `100` *through end*

DESCRIPTION

DEL deletes a range of BASIC lines from RAM. The BASIC interpreter "repacks" the BASIC program to utilize the deleted area. If the "DEL-" format is used, the entire program is deleted from memory. If the "line#-line#" format is used, all lines including the start and end lines are deleted. If the "-line#" format is used, all lines from the beginning of the program through the specified end number are deleted. If the "line#-" format is used, all lines from the specified start number through the end of the program are deleted. DELETE may be used to delete lines from the command mode for program editing purposes, or to delete program lines "dynamically" to release portions of BASIC programs that are no longer needed to create room for variables.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II •
I, Disk •
II •
III, LVL I
III, LVL III •
III, Disk •
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

`DELETE` *line#-line# (in command mode)*
`DELETE` *line#*
`DELETE` *-line#*
line# `DELETE` *line#-line#*

EXAMPLES

`DELETE 100-` *delete lines* `100` *through end*

DESCRIPTION

DELETE deletes a range of BASIC lines from RAM. The BASIC interpreter "repacks" the BASIC program to utilize the deleted area. If the "line#-line#" format is used, all lines including the start and end lines are deleted. If the "-line#" format is used, all lines from the beginning of the program through the specified end line number are deleted. If the "line#" format is used, the specified line number is deleted. DELETE may be used to delete lines from the command mode for program editing purposes, or to delete program lines "dynamically" to release portions of BASIC programs that are no longer needed to create more room for variables.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# `DIM` *name(dim1)*
line# `DIM` *name(dim1,dim2)*
line# `DIM` *name(dim1,dim2,...dimk)*

EXAMPLES

`1000 DIM A%(10,40)` *11 by 41 int array*

DESCRIPTION

`DIM` is used to allocate space for a `BASIC` array. The name parameter names an integer, single precision, double precision, or string array (numeric or string in the Color Computer). The name must adhere to the name conventions for the variable type involved. The dimensions are one less than the number of elements for each dimension of the array. The `DIM` statement only names and allocates the array; it does not initialize it to any value, although the elements are zeroed on power up automatically. Elements within the array are accessed by using the element number with the array name. The first element of a two-dimensional array might be A(0,0), the second A(0,1), and so forth. The last element in the array has the element numbers defined in the `DIM` statement. Each array element requires the same memory that a variable of the same type would require.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

`DIR`
`DIR`*drive#*

EXAMPLES

`DIR0`

DESCRIPTION

`DIR` displays the disk directory of the disk drive number specified. If the drive number is not used, `DIR` will display the directory of the current disk drive (last specified by `DRIVE`) or drive 0, the default drive number if `DRIVE` has not been used. The directory will be displayed with the file name, extension of the file (`BAS`, `BIN`, `DATA`, or other user- or system-specified extension), file type (0=`BASIC` data file, 1=`BASIC` data file, 2=machine-language file, 3=editor source file), file format (A=ASCII, B=binary), and file length in granules (2304 bytes). A typical display line might be:

```
ACCTS DATA  1 B 5
```

indicating file `ACCTS/DATA`, a `BASIC` data file in binary that is 5 granules or 11520 bytes long.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# DRAW *"string"*

EXAMPLES

```
1000 DRAW ''BM128,96;M0,0;M255,2''
```

DESCRIPTION

The DRAW command is used to draw a series of connected line segments in various lengths and directions. The line segments may be drawn in 8 directions in any length. The "string" parameter specifies a string of DRAW subcommands, each defined by a single text character. To draw a line of n pixels up, 45 degrees, right, 135 degrees, down, 215 degrees, left, or 325 degrees, use the text strings "Un;", "En;", "Rn;", "Fn;", "Dn;", "Gn;","Ln;", or "Gn;", where n is the number of pixels. To move to any x,y coordinate, use the text string "Mx,y;" where x and y are 0-255 and 0-191, respectively. Precede x and y with "+" or "-" for moves relative to the current position. Use "B" after the M or "B;" at any time for a "blank" line. Use "N" before the motion command for a "no update" of the position. Use "Cn;" to change color. Use "Ax;" for rotates of 0, 90, 180, or 270 degrees (x=0,1,2,3). Scale the draw by "Sx;" where x equals a scale factor of 1 through 62. Execute a substring by "X(string);".

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

DRIVE *drive#*

EXAMPLES

```
DRIVE 1
```

DESCRIPTION

DRIVE is a Color Computer Disk BASIC command. It is only used on systems with more than one drive to change the "default" disk drive number. The default drive number is used when the drive number is not specified in a filename (the standard filename format is name/extension:drive number).

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk •

FORMAT

line# DSKI$ *drive#,track,sector,string var 1,string var 2*

EXAMPLES

1000 DSKI$ 0,12,3,A$,B$ *drive 0,track 12, sector 3*

DESCRIPTION

DSKI$ is a Color Computer Disk BASIC command that permits direct access of a specified physical location on disk. It is used to process special files created by the system user or to process disk contents without using disk "file manage". The drive# parameter specifies the drive, the track parameter one of the diskette tracks (0 through 34), the sector number one of the sectors within the track (0 through 17). The two string variables receive the 256 bytes of data from the track, sector. String variable 1 receives the first 128 bytes from the sector, while string variable 2 receives the second 128 bytes. Data from the disk may or may not represent valid ASCII characters, depending upon the data output to the disk.

RELATED COMMANDS

DSKO$

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk •

FORMAT

DSKINI*drive#*

EXAMPLES

DSKINI0

DESCRIPTION

DSKINI is a Color Computer Disk BASIC command that "formats" a diskette in the specified drive number. The formatting process prepares the diskette for receiving data files and is a necessary process before doing any BASIC disk operations.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# DSKO$ *drive#,track,sector,string 1,string 2*

EXAMPLES

1000 DSKO$ 0,12,3,A$,B$ *drive 0,track 12, sector 3*

DESCRIPTION

DSKO$ is a Color Computer Disk BASIC command that permits direct access of a specified physical location on disk. It is used to create special files defined by the system user. The drive# parameter specifies the drive, the track parameter one of the diskette tracks (0 through 34), the sector number one of the sectors within the track (0 through 17). The two string variables define the 256 bytes of data to be output to the track, and sector. String variable 1 defines the first 128 bytes for the sector, while string variable 2 defines the second 128 bytes. Literal strings may be used in either case. Data in the variables may or may not represent valid ASCII characters, depending upon the data to be output. DSKI$ is normally used to input the disk data output by DSKO$.

RELATED COMMANDS

DSKI$

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...x.xxxxEyy

EXAMPLES

```
1000 A=1.1112E-5
1010 ZZ!=3.567E+34
```

DESCRIPTION

E is used to denote scientific notation for single-precision numbers. The format consists of a fraction or mixed number, followed by a D, followed by a power of ten. The power of ten may be positive (plus sign or no sign) or negative (minus sign). The fraction or mixed number may be any number of decimal digits up to 7, with the decimal point located anywhere within the digits. The decimal point is optional. The variable associated with the E format must be a single-precision variable. This is the default condition for BASIC variables and no "!" suffix is necessary unless the variable name falls in a DEFDBL or DEFSTR range.

RELATED COMMANDS

!, DEFSNG

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

`EDIT` *line# (in command mode)*

`EDIT.` *(except Color Computer)*

EXAMPLES

`EDIT 1000` *edit line #* `1000`

`EDIT.` *edit last line entered, altered, or in error*

DESCRIPTION

EDIT is a command mode command that invokes the BASIC interpreter Edit mode. The edit mode is used to modify BASIC program lines by adding, deleting, or modifying characters to the line. Any existing line number may be specified in the EDIT command. After the EDIT command has been given, the BASIC interpreter will display the line number and will position the cursor to the first character of the line. Subsequent Edit mode commands will allow editing of the line. To get out of the Edit mode, press ENTER. The "EDIT." format displays the last line entered, altered, or in which an error occurred. Entering the Edit mode automatically clears all variables. If BASIC encounters a syntax error during program execution, it automatically enters the Edit mode for the erroneous line. Entering "Q" will allow you to Quit the Edit mode and examine variables and program conditions.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: A keypress

EXAMPLES

`1000 FOR I=1 TA J-` *(pressing A cancels changes and restarts the Edit)*

DESCRIPTION

The Edit mode is entered by the EDIT line# command. The A subcommand is used to cancel all changes to the line that have been made and to restart the Edit at the beginning of the line. The A subcommand differs from the Q subcommand in that the Q subcommand cancels changes and Quits the Edit mode, while the A subcommand cancels changes but keeps the Edit mode in force. In the example above, the result would have been

```
1000 FOR I=J TA J
1000 -
```

The line can now be reedited with the proper changes.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: Backspace keypress (backspace is left arrow)
Edit Mode: nBackspace keypress

EXAMPLES

`1000 FOR I=1 TO -` *(pressing 5 and Backspace backspaces to the left 5 characters on the line)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. To backspace the cursor to the left one character position, press Backspace (left arrow). Jo backspace to the left more than one character position, enter a number of 1 through n and press Backspace. In the example above, 5 was entered, followed by Backspace; this positioned the cursor 5 character positions to the left. The 5 characters previously displayed were unaltered but erased from the display. Backspace can be used to space back along the line until the proper place is found to insert, delete, or modify characters by the other Edit Mode subcommands.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: C keypress
Edit Mode: nC keypress

EXAMPLES

`1000 FOR I=1 TO -` *(pressing 5 and C begins change operation for next 5 characters)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. The C subcommand is used to change 1 or more characters to new characters. To change the current character at the cursor position, press C followed by the new character. To change n additional characters, enter a number of 1 through n and press C. Then type the characters to replace the number specified. In the example above, 5 was entered, followed by C. If (K-5) was then entered, the new line up to that point would read

```
1000 FOR I=1 TO (K-5)-
```

The number of characters for the change must be exactly equal to the number replaced.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: D keypress
Edit Mode: nD keypress

EXAMPLES

`1000 FOR I=1 TO -` *(pressing 5 and D deletes next 5 characters)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. The D subcommand is used to delete 1 or more characters. To delete the current character at the cursor position, press D. The character deleted will be displayed bracketed by exclamation points. To change n additional characters, enter a number of 1 through n and press D. The characters deleted will be displayed bracketed by exclamation points. In the example above, 5 was entered, followed by D. The display would show:

```
1000 FOR I=1 TO !(K-5)!-
```

The characters (K-5) would have been deleted from the line.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: E keypress

EXAMPLES

`1000 FOR I=1 TO J-5 STEP -` *(press E)*

DESCRIPTION

The Edit mode is entered by the `EDIT` line# command. Pressing the E key while in the `Edit` Mode records all changes made while in Edit mode and returns to the `BASIC` interpreter command mode. E is not active while in any Insert mode such as I, X, or H. E is logically equivalent to pressing ENTER.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: ENTER keypress

EXAMPLES

`1000 FOR I=1 TO J-5 STEP -` *(press ENTER)*

DESCRIPTION

The Edit mode is entered by the `EDIT` line# command. Pressing the ENTER key while in the `EDIT` mode records all changes made while in Edit mode and returns to the `BASIC` interpreter command mode.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

Edit Mode: ESC keypress

EXAMPLES

`1000 FOR I=1 TO -` *(pressing ESC resets the Insert mode)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. Text may be inserted by the I, X, or H subcommands. While in the edit portion of these subcommands, characters are entered until the ESC key is pressed. The Insert submode is then ended. ESC should be pressed at any time to "reset" the current Edit mode to a known condition.

RELATED COMMANDS

Edit Mode Subcommands I, H, X

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: H keypress

EXAMPLES

`1000 FOR I=-` *(pressing H deletes remainder of line and invokes the Insert mode)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. To delete the remainder of the line from the current cursor position, press H. This "Hacks off" the remainder of the line and invokes the Insert mode. In the example above, pressing H and then entering "2 TO K-6" would have resulted in the following line:

```
1000 FOR I=2 TO K-6-
```

At this point the Insert mode would still be in force and additional characters could be added to the end of the line. To terminate the Insert mode, press SHIFT, up arrow together, or press ENTER. ENTER enters the current changes and returns to the command mode, while SHIFT, up arrow terminates the Insert mode but keeps the Edit mode active.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: I keypress

EXAMPLES

`1000 FOR I=1 TO -` *(pressing I enters Insert submode)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. To insert characters at any point press I. All characters entered from that point until the SHIFT, up arrow keys were pressed simultaneously would be entered into the line. In the example above, if the original line was "`1000 FOR I=1 TO 100`", entering I followed by "J-" and then SHIFT, up arrow would result in a line consisting of:

```
1000 FOR I=1 TO J-
```

The SHIFT, up arrow would not terminate the Edit of the line; the cursor would be positioned after the last character inserted and the remainder of the line would not be visible. Pressing the ENTER key will also terminate the Insert.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: Kc keypress
Edit Mode: nKc keypress

EXAMPLES

`1000 -` *(pressing 2, K, and : searches for the second occurrence of the character ":" and kills all characters to that point)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. The K subcommand is used to search for the first or "nth" occurrence of a single character and to delete all characters preceding the search character from the current cursor position. To search for the first occurrence of a character, press K followed by the search character. The cursor will move to the right until positioned over the character and delete all characters from the cursor position to that point. The deleted text will be displayed bracketed by exclamation points. The search character will not be displayed. To search for the nth occurrence of a character, enter a number from 1 to n, enter a K, and enter the search character. The cursor will be positioned over the nth occurrence of the character with a similar delete action.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: L keypress

EXAMPLES

`1000 FOR I=-` *(pressing L displays remainder of line)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. To display the remainder of the line, press L. The remainder of the line will be displayed and a new line will be started with the cursor positioned on the first character of the new line. In the example above, the result would have been

```
1000 FOR I=1 to J-5 STEP 3
1000 -
```

The Edit Mode L subcommand lets you see the remainder of the line without having to space along the line. The L subcommand is not active while in an insert mode such as I, X, or H.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: Q keypress

EXAMPLES

`1000 FOR I=1 TA J-` *(pressing Q cancels changes and Quits the Edit)*

DESCRIPTION

The Edit mode is entered by the EDIT line# command. The Q subcommand is used to cancel all changes to the line that have been made and to Quit the Edit. The Q subcommand differs from the A subcommand in that the Q subcommand cancels changes and Quits the Edit mode, while the A subcommand cancels changes but keeps the Edit mode in force. In the example above, the result would have been

```
1000 FOR I=J TA J
```
(BASIC command mode)

The Q subcommand is used when changes have been erroneously made to a BASIC program line.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: Sc keypress
Edit Mode: nSc keypress

EXAMPLES

`1000 -` *(pressing 2, C, and O searches for the second occurrence of the letter O)*

DESCRIPTION

The Edit Mode is entered by the EDIT line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. The S subcommand is used to search for the first or "nth" occurrence of a single character. To search for the first occurrence of a character, press S followed by the search character. The cursor will move to the right until positioned over the character. The character will not be displayed. To search for the nth occurrence of a character, enter a number from 1 to n, enter an S, and enter the search character. The cursor will be positioned over the nth occurrence of ther character. The line up until the nth occurrence will be displayed. If the character is not found in the search, the entire line will be displayed with the cursor positioned at the end.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	●
I, Disk	●
II	●
III, LVL I	
III, LVL III	●
III, Disk	●
CC, BASIC	
CC, Ext BASIC	●
CC, Disk	●

FORMAT

Edit Mode: SHIFT, up arrow

EXAMPLES

`1000 FOR I=1 TO -` *(pressing SHIFT, up arrow resets the Insert mode)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. Text may be inserted by the I, X, or H subcommands. While in the edit portion of these subcommands, characters are entered until the SHIFT, up arrow keys are pressed simultaneously. The Insert submode is then ended. SHIFT, up arrow should be entered at any time to "reset" the current Edit mode to a known condition.

RELATED COMMANDS

Edit Mode Subcommands I, H, X

SYSTEM

I, LVL I	
I, LVL II	●
I, Disk	●
II	●
III, LVL I	
III, LVL III	●
III, Disk	●
CC, BASIC	
CC, Ext BASIC	●
CC, Disk	●

FORMAT

Edit Mode: Space-Bar press
Edit Mode: nSpace-Bar press

EXAMPLES

`1000 FOR I=1 TO -` *(pressing 5 and space bar displays and spaces 5 additional characters on the line)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. To display an additional character, press Space-Bar. To display n additional characters, enter a number of 1 through n and press Space-Bar. In the example above, 5 was entered, followed by Space-Bar; this displayed 5 additional characters on the line and positioned the cursor after the 5 additional characters. Space-Bar can be used to space along the line until the proper place is found to insert, delete, or modify characters by the other Edit Mode subcommands.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

Edit Mode: X press

EXAMPLES

`1000 -` *(pressing X displays remainder of line and invokes the Insert mode)*

DESCRIPTION

The Edit Mode is entered by the `EDIT` line# command. While in the Edit Mode, the current line is displayed in whole or in part. The cursor is positioned somewhere along the line. To display an additional character, press Space-Bar. To display the remainder of the line and position the cursor to the end of the line in the Insert mode, press X. In the example above, pressing X would have displayed

```
1000 FOR I=1 TO J-5 STEP 3-
```

At this point the Insert mode would be in force and additional characters could be added to the end of the line. The X command is an "Extend Line" command and is used for that purpose. To terminate the Insert mode, press SHIFT up arrow together, or press ENTER. ENTER enters the current changes and returns to the command mode, while SHIFT up arrow terminates the Insert mode but keeps the Edit mode active.

RELATED COMMANDS

Edit Mode Subcommands

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# `END`

EXAMPLES

`1000 END` *stops execution and returns to the command mode*

DESCRIPTION

`END` determines an end point of the `BASIC` program. When encountered by the `BASIC` interpreter, `END` causes the interpreter to stop program execution and return to the command mode. There may be any number of `END`s in the `BASIC` program. It does not define the physical end of the program, but is only relevant during program execution.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	●
II	●
III, LVL I	
III, LVL III	
III, Disk	●
CC, BASIC	
CC, Ext BASIC	
CC, Disk	●

FORMAT

line#...EOF(buf#)...

EXAMPLES

```
1000 IF EOF(1) THEN CLOSE(1):GOTO
2000
```

DESCRIPTION

EOF is a Disk BASIC function that indicates whether the "end-of-file" of a disk file has been reached. It is normally used during a disk read operation to test for the read of the last data from the file. Two types of reads might be done. In one type, the user knows exactly how many records are in a disk file and reads that exact number. In the second type, the user tests for EOF to determine when all of the data has been read. In the EOF case, a 0 is returned when more data remains in the file, and a -1 is returned when all data has been read and an EOF condition exits. The EOF is used in this context as a "logical" function which specifies a true/false condition.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	●
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line# ...expression*EQV*expression...*

EXAMPLES

```
1000 C=A EQV B
```

DESCRIPTION

EQV is a logical or bit manipulation operator that processes two operands in similar fashion to the more common AND or OR. EQV compares both operands (constants, variables, or expressions on a bit by bit basis. For each bit position, the result bit is a 1 when both bits are the same. `0 IMP 0=1; 0 IMP 1=0; 1 IMP 0=0;` and `1 IMP 1=1`. EQV is the inverse of the XOR function. The expressions are converted to 16-bit integers and then compared on a bit basis. If A is binary 01010000 and B is 00111111, above, then C is 10010000.

RELATED COMMANDS

XOR

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# `ERASE` *array1,arrray2,array3*

EXAMPLES

`1000 ERASE XX,A%,A$` *erase three arrays*

DESCRIPTION

ERASE is used to "de-allocate" one or more arrays. When ERASE is executed, the specified arrays are removed from RAM space, and the area allocated for the arrays is released to the free memory area. ERASE is the opposite of DIM. Arrays deleted in an ERASE may be redimensioned. ERASE removes the entire array and cannot be used to remove one or a few entries of the array.

RELATED COMMANDS

DIM

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line#...`ERL...`

EXAMPLES

`1000 IF ERL=2000 THEN STOP` *stop if invalid read in line* 2000

DESCRIPTION

ERL is a special error-processing function which returns the line number in which an error occurred. The ERL is normally used within an error-processing routine defined by the line number in an ON ERROR GOTO command. When any error occurs and the user error-handling mode is in force, the error-processing routine takes suitable actions for the error, such as displaying the type of error, line number, and corrective action. The ERL allows the error-processing routine to determine the line number and therefore further information about the manner of error and action to take. If a program error has occurred since power up, ERL returns the line number of the last error. If an error occurred in the command mode (such as entering LLLIST), 65535 is returned as the ERL argument to signify that no line number was involved.

RELATED COMMANDS

ERR,ERROR, ON ERROR GOTO, RESUME

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*ERR...

EXAMPLES

`1000 IF ERR/2+1=4 THEN STOP` *stop if out of data*

DESCRIPTION

ERR is a special error-processing function which returns the error code for the error that just occurred. ERR is normally used within an error-processing routine defined by the line number in an ON ERROR GOTO command. When any error occurs and the user error-handling mode is in force, the error-processing routine takes suitable actions for the error, such as displaying the type of error, line number, and corrective action. The ERR allows the error-processing routine to determine the type of error and therefore define the manner of error and action to take. The expression ERR/2+1 is used to find the true error code for the Models I and III.

RELATED COMMANDS

ERL, ERROR, ON ERROR GOTO, RESUME

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ERROR *code*

EXAMPLES

`1000 ERROR 4` *simulate out of data error*

DESCRIPTION

ERROR is used to simulate an error condition. ERROR is primarily used to test a user error-processing routine. The error-processing routine is established by an ON ERROR GOTO command with appropriate error handing code.

RELATED COMMANDS

ON ERROR GOTO

SYSTEM	
I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...ERR$...

EXAMPLES

```
1000 PRINT "ERROR: ";ERR$
```

DESCRIPTION

ERR$ returns a text string containing the number and description of the TRSDOS error related to the latest BASIC disk error. BASIC normally displays a "DISK I/O" error indication. ERR$ is a way of further defining the error in TRSDOS. ERR$ would normally be used in BASIC error-handling routines to notify the user of errors and to determine some corrective action. If no TRSDOS error occurred, ERR$ returns a null string.

RELATED COMMANDS

ON ERROR GOTO

SYSTEM	
I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

EXEC
EXEC *address*

EXAMPLES

EXEC *execute last loaded machine-language program*

DESCRIPTION

EXEC causes a transfer to the last CLOADM address or to the specified address value. EXEC is used primarily after a CLOADM command to transfer control to the machine-language file, assumed to be a major program (one not generally interfacing to BASIC via the USR command). EXEC may also be used in the "EXEC address" format to transfer control to any machine-language code at any time while in the command mode. The address parameter specifies the starting address for execution.

RELATED COMMANDS

CLOADM

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...`EXP`*(expression)...*

EXAMPLES

```
1000 A=EXP(X)
```

DESCRIPTION

`EXP` is the inverse of the `LOG` function. It returns the natural exponential of X, or e (2.718...) to the X power. Natural logarithms and exponentials are used in a variety of mathematical and scientific applications.

RELATED COMMANDS

`LOG`

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# `FIELD` *buf#,n AS name1,n AS name2,...,n AS namen*

EXAMPLES

```
1000 FIELD 1,20 AS LNAME$,20 AS
FNAME$,40 AS ADDR$
```

DESCRIPTION

`FIELD` is used to define fields of specified length within a random-file buffer. Fields are subdivisions of a record. Each field has a name specified in the field statement. The field name may be used in `LSET`, `RSET` or other commands to easily store or retrieve character data from the record without having to specify the relative location of the data in numeric form. It would be much more convenient to reference "`FNAME`" for "first name" than the 20th through 39th characters in a record, for example. The buf# parameter defines the buffer number to be used when referencing data. The buffer number is associated with a file by the `OPEN` command. The n parameters define the length of the field in characters. The name parameters define a field string variable name. (`DUMMY$` can be used to "space over" characters.) The total number of characters used for the fields must equal the record length defined in the `OPEN`.

RELATED COMMANDS

`LSET`, `RSET`

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

FILES *number of bufs,buffer size*
line# FILES *number of bufs,buffer size*

EXAMPLES

FILES 3,256 *reserve 3 bufs of 256 bytes*

DESCRIPTION

FILES specifies how many disk buffers to reserve in memory and how large the buffers should be. The buffer size parameter is optional; if not used, a buffer size of 256 bytes is used. Disk BASIC uses buffers to assemble records on output to disk and to read in sectors of the disk on input. Sectors are 256 bytes long, and this is the normal length for RAM buffers. If FILES is never specified, two buffers of 256 bytes are assumed.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*FIX*(expression)...*

EXAMPLES

1000 REM FIND INTEGER PORTION OF X
1010 IN=FIX(X) *put integer portion in* IN

DESCRIPTION

FIX finds the integer portion of a constant, variable, or expression. Unlike INT, it finds the true integer portion of a negative argument. The integer portions of +1.12, +100.45, 0, -5.567, and -999.999 are 1, 100, 0, -5, and -999, respectively. The argument must be within parentheses. The argument does not have to be an integer value (-32768 to +32767).

RELATED COMMANDS

INT

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# FOR *variable=expression* TO *expression*
STEP *expression*

EXAMPLES

```
1000 FOR I=0 TO 100 loop 101 times
2000 FOR I=7 TO 100 STEP 2 loop 47 times
3000 FOR I=101 TO 0 STEP -2 loop 51
times
```

DESCRIPTION

The FOR...TO...STEP commands, together with NEXT, set up and execute a program loop. The "variable" is executed from the starting value given in the expression 1 TO an ending value given in expression 2. The two start and end values may be constants, variables, or expressions. If no STEP size is given, the variable is incremented by one each time the loop is repeated, until the variable equals the end value. If a STEP size is given, the variable increments by the STEP size each time through the loop. The start and end values may be positive or negative. If the start is less than the end value, a STEP of a negative value is mandatory. A NEXT command later in the program defines the end of the loop and transfers control back to the FOR...TO...STEP statement for the next iteration of the loop. Any number of loops may be "nested".

RELATED COMMANDS

NEXT

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

FRE*(string)*
*line#...*FRE*(string)...*

EXAMPLES

```
1000 PRINT FRE(A$)
```

DESCRIPTION

FRE returns the amount of free string storage space available in bytes. In finding the amount of string storage, the BASIC interpreter "cleans up" the string storage area near the top of RAM to create the maximum free string space. The string storage area size was first specified in a CLEAR statement. If no CLEAR statement was encountered, 50 bytes of string storage space is automatically saved. The "string" parameter within parentheses is a "dummy" argument; the string variable specified has no significance. FRE is usually entered from the command mode, although it can be used within a BASIC program as a check on free string space. If the argument in FRE is numeric, FRE returns the total amount of free memory.

RELATED COMMANDS

CLEAR

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# ...FREE*(drive#)*...

EXAMPLES

```
PRINT FREE(1)
```

DESCRIPTION

FREE is a Color Computer Disk BASIC command that returns the number of free granules on the diskette for the specified disk drive. A granule is the minimum unit of disk drive space allocated by the BASIC "file manage" handler and is equal to 5 sectors, or 2304 bytes. FREE is used either in the command mode or embedded in a program to find the space remaining on a diskette for user programs or data.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# GET *buf#*
line# GET *buf#,rec#*

EXAMPLES

1000 GET 3,100   *get 100th record*

DESCRIPTION

GET is used to read a random-access file record from disk. A random-access file allows records to be read or written on a random basis (not in sequence). The GET permits either the next record in sequence or any record number of the file to be read into the buffer associated with the file. Prior to the GET, an OPEN with the "R" option must have been executed. The OPEN defines the filename and buffer associated with the file. The "GET buf#" form of GET reads in the current record, the number whose number is one higher than the last access. If no record has been read, this is the first record of the file. The second form of GET reads in the specified record defined by "rec#".

RELATED COMMANDS

PUT

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# GET*(x1,y1)-(x2,y2),array name,g*

EXAMPLES

```
1000 GET (0,0)-(50,50),AA,G  save
area in array AA
```

DESCRIPTION

The GET command is used in conjunction with the PUT command. GET stores any rectangular area on a graphics screen in a two-dimensional array. The PUT later retrieves the graphics data from the array and displays it in any other area of the graphics screen. GET/PUT can be used to save portions of a graphics screen or to create animation effects. The x1,y1 coordinates define one corner of the rectangle to be stored in the array; The x2,y2 define the opposing corner. The x1,x2 and y1,y2 values are in "high-resolution" graphics coordinates of 0-255 and 0-191, respectively. The "array name" is the name of a two-dimensional array previously defined by a DIM statement. In general, the array size must be equal to the dimensions of the graphics area to be stored, although certain space-saving tricks may be used. The g option is "G"; if used, full graphic detail is saved in the array.

RELATED COMMANDS

PUT

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# GOSUB *line#*

EXAMPLES

```
1000 REM DO SEARCH SUBROUTINE
1010 GOSUB 12000
1020 REM RETURN HERE AFTER
SUBROUTINE
```

DESCRIPTION

GOSUB is used to "call" a subroutine. A subroutine is any set of BASIC statements that is used repeatedly. Making the statements a subroutine in one spot rather than repeating the code when required saves RAM space. The GOSUB causes the BASIC interpreter to branch to the line number specified after the GOSUB. Unlike the GOTO, the GOSUB action saves the return point after the GOSUB. After the subroutine has been executed, the last statement of the subroutine, a RETURN, returns control to the statement after the GOSUB. In the example above, the subroutine at line 12000 would be executed; it could consist of from one to many statements. The last statement, however, is a RETURN, which causes a return to line number 1020. Subroutines may be "nested" in many levels. One subroutine may call another by a GOSUB, which may call yet another, etc.

RELATED COMMANDS

ON...GOSUB, RETURN

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# GOTO *line#*
GOTO *line#*

EXAMPLES

1000 GOTO 2000 *transfers control to line # 2000*

GOTO 2000 *continues at line* 2000

DESCRIPTION

GOTO is used in BASIC programs to transfer control from one statement to another. It is the normal way of "unconditionally branching" in the program. Any number of GOTOs may be used in a program. When a GOTO is executed, no record of where the GOTO occurred is kept by the BASIC interpreter, unlike a GOSUB. When a GOTO is used in the command mode, the BASIC program continues from the specified line number with all variables and BASIC parameters intact. The GOTO in this use may be used in lieu of a CONT (continue) to restart the program at any point.

RELATED COMMANDS

CONT, GOSUB

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*HEX$*(expression)...*

EXAMPLES

1000 PRINT HEX$(A) *find hex value of A*

DESCRIPTION

HEX$ is a special function that will convert a constant, variable, or expression to a string that represents the hexadecimal value of the argument. HEX$(1000), for example, will be converted to the string "3E8". Hexadecimal notation is used primarily for machine-language operations in specifying addresses, instruction codes, and data values.

RELATED COMMANDS

&H

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# IF *true/false expression* THEN *action*

EXAMPLES

```
1000 IF A<25 THEN A=25  test A
1010 IF (A=3 OR B=6) THEN GOTO 4000
```

DESCRIPTION

The IF...THEN command is used to test a true/false condition and to take some action if the result is true. If the result is not true, the next statement in sequence is executed. The true/false expression may contain any relational operators, such as test for equality (A=B), sense (A<B), string comparisons (A$<B$), and others. Constants, variables, or expressions may be used in the true/false expression in any mixture. The action to be taken if the true/false expression is true may be any one statement action, such as "THEN PRINT A", or "THEN A=(3.66*I-2)". The THEN is not necessary in the case of a transfer to a line# such as "THEN GOTO 3000". If multiple statements are on a single line after the THEN, all statements after the THEN will be executed if the true/false expression is true. The line "1000 IF A<2 THEN A=1:B=23:PRINT C" will result in A set equal to 1, B set equal to 23 and C being printed if A is less than 2.

RELATED COMMANDS

IF...THEN...ELSE

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# IF *true/false expression* THEN *action* ELSE *action*

EXAMPLES

```
1000 IF A<2 THEN A=A+4 ELSE A=A+5
1010 IF B=(I+37) THEN C=5 ELSE IF
B=(I+38) THEN C=6
```

DESCRIPTION

The IF...THEN...ELSE command is used to test a true/false expression and to take the THEN action if the statement is true and the ELSE action if the statement is false. The true/false expression may use any relational operators as in "IF A=2", "IF A<2", "IF A$<B$". If the true/false expression is true, the THEN action is taken and the ELSE action disregarded. The THEN action may be a single statement action of any type. If the true/false expression is false, the ELSE action is taken and the THEN action disregarded. The THEN action may be any single statement action. A line number may be used without a GOTO following the THEN or ELSE. "Nested" IF...THEN...ELSE commands may be used as shown in the example above. If multiple statements follow the ELSE, then all actions up to the end of the line are taken in the false condition.

RELATED COMMANDS

IF...THEN

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	●
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...expression `IMP`*expression...*

EXAMPLES

```
1000 C=A IMP B
```

DESCRIPTION

IMP is a logical or bit manipulation operator that processes two operands in similar fashion to the more common AND or OR. IMP compares both operands on a bit by bit basis. For each bit position, the result bit is a 1 unless the bit of the first operand is a 1 and the bit of the second operand is a 0. 0 IMP 0=1; 0 IMP 1=1; 1 IMP 0=0; and 1 IMP 1=1. The expressions are converted to 16-bit integers and then compared on a bit basis. If A is binary 01010000 and B is 00111111, above, then C is 10111111.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	●
I, Disk	●
II	●
III, LVL I	
III, LVL III	●
III, Disk	●
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

line#...`INKEY$...`

EXAMPLES

```
1000 IF INKEY$<>"" THEN GOTO
2000  go if key press
```

DESCRIPTION

INKEY$ is a special string function that allows you to read the keyboard at "real-time" rates. If no key is being pressed on the keyboard, INKEY$ is set equal to a "null" string of zero length, defined by "". If a key is being pressed, INKEY$ is set equal to the current key press on the keyboard for a brief period. If the key is not released, INKEY$ is shortly set equal to a "null" string. If one key is being depressed and a second is pushed, INKEY$ is set equal to the second key (for a brief period). Successive pushes of the same key result in short bursts where INKEY$ is set equal to the key character interspersed with longer periods where INKEY$="". INKEY$ can be used in a loop to test for key presses at real-time rates. The following code builds up a string of keypushes:

```
1000 B$=INKEY$
1010 IF B$="" THEN GOTO 1000 ELSE
A$=A$+B$: GOTO 1000
```

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# INP*(port)*

EXAMPLES

```
1000 A=INP(255)
```
read cassette on Model I/III

DESCRIPTION

INP inputs a one-byte value from a system input/output port. The Model I/II/III uses input/output ports for certain system devices such as cassette and RS-232-C operations. The INP is a BASIC command that will enable the user to directly read these I/O ports. The port parameter is an address value of 0 through 255 that defines the port address. It must be within parentheses. INP returns a one-byte (8-bit) value representing input data on the specified port address.

RELATED COMMANDS

OUT

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# INPUT *item list*

EXAMPLES

```
1000 INPUT A$,EN,AG
```
input name,number,age

DESCRIPTION

INPUT is used to enter data from the keyboard. Data is entered as a list of items. For each item in the data list, INPUT accepts a numeric or string variable. Entries may be entered one at a time from the keyboard or all entries may be entered with each individual item separated by commas. The type of entry must match the data item type - numeric items cannot include text. If an invalid item type is entered, a "REDO" message is output. BASIC prompts the user by a "?" when INPUT is expected. If more than one item is in the INPUT list and not all entries have been entered when the ENTER key is pushed, BASIC indicates that more items are expected by "??". Entering more items than there are in the list causes an "?EXTRA IGNORED" message.

RELATED COMMANDS

None

SYSTEM

I, LVL I	●
I, LVL II	●
I, Disk	●
II	●
III, LVL I	●
III, LVL III	●
III, Disk	●
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

line# `INPUT` *"text";item list*

EXAMPLES

```
1000 INPUT ``ENTER
NAME,#,AGE'';A$,EN,AG
```
input name,number,age

DESCRIPTION

INPUT...; is identical to the normal INPUT statement except that a message is displayed before the INPUT. The text of the message is enclosed by double quotes and separated from the item list by a semicolon. INPUT is used to enter data from the keyboard. Data is entered as a list of items. For each item in the data list, INPUT accepts a numeric or string variable. Entries may be entered one at a time from the keyboard or all entries may be entered with each individual item separated by commas. The type of entry must match the data item type - numeric items cannot include text. If an invalid item type is entered, a "REDO" message is output. BASIC prompts the user when INPUT is executed by a "?". If more than one item is in the INPUT list and not all entries have been entered when the ENTER key is pushed, BASIC indicates that more items are expected by "??". Entering more items than there are in the list causes an "?EXTRA IGNORED" message.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	●
II	●
III, LVL I	
III, LVL III	
III, Disk	●
CC, BASIC	
CC, Ext BASIC	
CC, Disk	●

FORMAT

line# `INPUT#` *buf#,item list*

EXAMPLES

```
1000 INPUT#3,A,B,C$
```
input from disk file

DESCRIPTION

INPUT# is used to input a list of items from a sequential file on disk. It is similar to the keyboard INPUT statement except that the data items are read from a disk file. The disk file must have been previously OPENed; the OPEN associates the buf# parameter with a sequential disk file. Normally the data items have been output to the disk file with a PRINT# statement. The item list must follow the same sequence as the items in the disk file; if two numeric items are followed by one string item, then the three variables read must be numeric, numeric, string. Data in sequential files is written onto disk as a succession of ASCII characters. Even numeric data is output as a string of characters. The INPUT# reads in the character data, detects the terminators between data items, and converts each item to the proper type for the item list. Blanks and the ENTER character generally serve as terminators between numeric data items, while commas separate string variables.

RELATED COMMANDS

PRINT#

SYSTEM

I, LVL I	•
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# `INPUT#`,*item list*

EXAMPLES

`1000 INPUT#,A,B,C$` *input from cassette*

DESCRIPTION

`INPUT#` is used to input a list of items from a cassette file. It is similar to the keyboard `INPUT` statement except that the data items are read from a cassette file. Normally the data items have been output to the cassette file with a `PRINT#` statement. The item list must follow the same sequence as the items in the cassette file; if two numeric items are followed by one string item, then the three variables read must be numeric, numeric, string. Data in cassette files is written as a succession of ASCII characters. Even numeric data is output as a string of characters. The `INPUT#` reads in the character data, detects the terminators between data items, and converts each item to the proper type for the item list. Blanks and the ENTER character generally serve as terminators between numeric data items, while commas separate string variables.

RELATED COMMANDS

`PRINT#`

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# `INPUT#-1`,*item list*

EXAMPLES

`1000 INPUT#-1,A,B,C$` *input from cassette*

DESCRIPTION

`INPUT#-1` is used to input a list of items from a cassette file. It is similar to the keyboard `INPUT` statement except that the data items are read from a cassette file. Normally the data items have been output to the cassette file with a `PRINT#-1` statement. The item list must follow the same sequence as the items in the cassette file; if two numeric items are followed by one string item, then the three variables read must be numeric, numeric, string. Data in cassette files is written as a succession of ASCII characters. Even numeric data is output as a string of characters. The `INPUT#-1` reads in the character data, detects the terminators between data items, and converts each item to the proper type for the item list. Blanks and the ENTER character generally serve as terminators between numeric data items, while commas separate string variables.

RELATED COMMANDS

`PRINT#-1`

SYSTEM

I, LVL I
I, LVL II •
I, Disk •
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line# `INPUT#-2`,*item list*

EXAMPLES

`1000 INPUT#-2,A,B,C$` *input from cassette*

DESCRIPTION

`INPUT#-2` is used to input a list of items from a cassette file. It is identical to `INPUT#-1` except that the cassette file is on the second cassette drive. It is similar to the keyboard `INPUT` statement except that the data items are read from a cassette file. Normally the data items have been output to the cassette file with a `PRINT#-` statement. The item list must follow the same sequence as the items in the cassette file; if two numeric items are followed by one string item, then the three variables read must be numeric, numeric, string. Data in cassette files is written as a succession of ASCII characters. Even numeric data is output as a string of characters. The `INPUT#-1` reads in the character data, detects the terminators between data items, and converts each item to the proper type for the item list. Blanks and the ENTER character generally serve as terminators between numeric data items, while commas separate string variables.

RELATED COMMANDS

`PRINT#-1,PRINT#-2`

SYSTEM

I, LVL I
I, LVL II
I, Disk
II •
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line# ...`INPUT$`*(length,buf#)...*

EXAMPLES

`1000 A$=INPUT$(10,3)` *input 10 characters from disk*

DESCRIPTION

`INPUT$` is a function that specifies the number of characters that will be read from a sequential disk file. It is somewhat similar to `LINE INPUT#` except that the input string is terminated by a number of characters rather than the ENTER key. The length parameter is a value from 1 through 255. The buf# is the number of the sequential file input buffer specified in the `OPEN` statement associated with the file name. When `INPUT$` is executed, `BASIC` will wait until the specified number of characters are read from the disk file and then return all characters as a string. All characters read will be returned, including those that would normally be delimiters, such as commas. `1000 A$=INPUT$(10,3)`, for example, would specify that A$ would be set equal to the next 10 characters input from the disk file associated with buffer 3 and that the next line would not executed until those 10 characters were input.

RELATED COMMANDS

`LINE INPUT#`

SYSTEM	
I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...`INPUT$`*(length)*...

EXAMPLES

`1000 A$=INPUT$(10)` *input 10 characters from keyboard*

DESCRIPTION

`INPUT$` is a keyboard function that specifies the number of keyboard characters that will be read. It is somewhat similar to `LINE INPUT` except that the input string is terminated by a number of characters rather than the ENTER key. The length parameter is a value from 1 through 255. When `INPUT$` is executed, `BASIC` will wait until the specified number of characters are typed and then return all characters as a string. All characters typed will be returned, including those that would normally be delimiters, such as commas. The characters input will not be displayed on the screen. `1000 A$=INPUT$(10)`, for example, would specify that A$ would be set equal to the next 10 characters input from the keyboard and that the next line would not executed until those 10 characters were input.

RELATED COMMANDS

`LINE INPUT`

SYSTEM	
I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#... `INSTR`*(string1,string2)*

line#... `INSTR`*(position,string1,string2)*

EXAMPLES

`1000 A=INSTR(A$,''ISS'')` *look for "ISS" in A$*

DESCRIPTION

`INSTR` is a function that searches for a substring within a larger string. The string1 and string2 parameters are string literals or variables. (String literals will be enclosed in quotes; string variables will have the "$" suffix or `DEFSTR` definition.) If the first format is used, `INSTR` will search for string2 in string1. If string2 is found within string1, the starting position of the first occurrence of string2 will be returned. If string2 is not found within string1, 0 will be returned. Positions of strings are numbered from 1 through the length of the string in characters. If the second format is used, the "position" parameter is a constant, variable, or expression that specifies the starting position for the search. In the example above, if A$="MISSISSIPPI", `INSTR` would set A to 2. The second occurrence of ISS would have to be found by specifying a position greater than 2.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*INT*(expression)...*

EXAMPLES

```
1000 REM POKE ADDRESS
1010 POKE I+1,INT(AD/256): POKE
I,AD-(INT(AD/256)*256)
```

DESCRIPTION

INT returns the integer portion of a positive number and the next highest integer for a negative number. The argument may be a constant, variable, or expression and must be within parentheses. For arguments of +1.12, +999.45, 0, -1.11, and -234.56, INT returns +1, +999, 0, -2, and -235, respectively. INT is commonly used to find the two bytes of a 16-bit address for POKEs of addresses as in the example above, or for rounding operations, as in

```
1000 FIND X ROUNDED TO 2 DEC PLACES
1010 XR=INT(X*100+.5)/100
```

INT should be used to find the integer portion of positive numbers only; FIX should be used when both positive and negative numbers are involved. The argument in INT may be any size.

RELATED COMMANDS

FIX

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*JOYSTK*(n)...*

EXAMPLES

1000 A=JOYSTK(3)   *get y coordinate of joystick 2*

DESCRIPTION

JOYSTK is a special function that reads the joystick value. (The optional joysticks must be connected to the joystick plugs on the back of the Color Computer.) The n parameter defines the position parameter to be read. Each of the two joysticks will return an "x" coordinate and a "y" coordinate. Arguments of n=0 and n=1 read the x and y coordinates from the left joystick, respectively. Arguments of 2 and 3 read the x and y coordinates from the right joystick. The value returned for any of the 4 positions is 0 through 63. The up and left positions are 0 and the down and right positions are 63. Intermediate positions are proportional, for example, the center position of a joystick is 32,32. JOYSTK(0) must first be returned before JOYSTK(1)-JOYSTK(3) can be read.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

KILL *"filename"*
line# KILL *"filename"*

EXAMPLES

KILL ''ACCOUNTS/BAS:1'' *kill accounts payable*

DESCRIPTION

KILL deletes a file on disk. It is identical to the TRSDOS KILL command except that it may be performed inside BASIC in the command or execution modes. (Always CLOSE an open file before executing a KILL; if this is not done, the disk contents may be destroyed.) The "filename" is a filespec for a BASIC program stored on disk; it conforms to the general requirements for filespecs - name, extension, password, and drive number. If no drive number is specified, KILL will delete the file from the first disk that contains the filename. (The order for the search is drive 0, 1, 2, and 3.)

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*LEFT$*(string,n)...*

EXAMPLES

1000 A$=LEFT$(B$,4) *get the first 4 characters of B$*
1010 C$=LEFT$(B$,I) *get the first I characters of B$*
1020 D$=LEFT$(B$,(I+2)) *get the first I+2 characters of B$*

DESCRIPTION

LEFT$ finds the last n characters of a given string. The n parameter may be 0 to 255. The "string" parameter is a previously defined string. If B$="HEROINE", for example, A$=LEFT$(B$,4) will set A$="HERO". If n is greater than the length of the specified string, LEFT$ will return the entire string. A$=LEFT$(B$,20), for example, returns A$="HEROINE". The n argument may be a constant, variable, or expression. LEFT$ may be used to process "substrings" where a large string is made up of a number of substrings concatenated together for ease of handling.

RELATED COMMANDS

MID$, RIGHT$

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*LEN*(string)*

EXAMPLES

```
1000 LA=LEN(A$)
```
find # of characters in A$

```
1010 LB=LEN(B$)
```
find # of characters in B$

DESCRIPTION

LEN finds the length in characters of a specified string. The length is the actual number of characters in the string, not counting string pointers. The "string" variable must be a valid string variable and may be a string expression such as A$+B$ or STRING$(5,"*"). LEN produces a numeric variable of 0 to 255 which can be used in string processing. IF A$="THE ONLY ISM FOR ME IS COMPUTERISM", then LEN(A$)=34.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# LET *variable=expression*

EXAMPLES

```
1000 LET A=1.2345E-10: LET
B=3.14159
```

DESCRIPTION

LET is used primarily for compatibility with older versions of BASIC. LET was used on older BASICs prior to setting a variable equal to a value or expression. On all TRS-80 BASICs, LET is optional and the variable may be set without the LET, as in

```
1000 A=1.2345E-10: B=3.14159
```

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# LINE*(x1,y1)-(x2,y2)*,PSET
line# LINE*(x1,y1)-(x2,y2)*,PRESET
line# LINE*(x1,y1)-(x2,y2)*,PSET,B
line# LINE*(x1,y1)-(x2,y2)*,PRESET,B
line# LINE*(x1,y1)-(x2,y2)*,PSET,BF
line# LINE*(x1,y1)-(x2,y2)*,PRESET,BF

EXAMPLES

```
1000 LINE (23,23)-
(100,100),PSET draw line
1010 LINE (200,150)-
(220,170),PRESET,BF erase filled-in box
```

DESCRIPTION

LINE is used to draw a line, box (rectangle), or filled-in box on the current graphics page. The x1,y1 and x2,y2 parameters specify two points on the graphics screen. The values used for x1 and x2 are 0 through 255. The values used for y1 and y2 are 0 through 191. The x and y ranges are for the highest resolution graphics mode. The ..,PSET form draws a line in the current foreground color between x1,y1 and x2,y2; the ..,PRESET form draws the line in the current background color. The ..,PSET,B and ..,PRESET,B forms draw the outline of a box in the current foreground and background color, respectively. The ..,PSET,BF and ..,PRESET,BF forms fill in the box with the current foreground or background color.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# LINE INPUT *string variable*
line# LINE INPUT *"text";string variable*

EXAMPLES

```
1000 LINE INPUT ''ENTER STREET,
CITY, STATE'';AD$
```

DESCRIPTION

LINE INPUT inputs a line of text entered from the keyboard. The input is terminated by an ENTER. All keyboard characters are entered as legitimate characters. LINE INPUT is unlike INPUT in that commas and other delimiters are treated as normal text characters and included as part of the result string. The "text" parameter is optional. If included, the text message is displayed just prior to the input operation. The resulting string variable includes all characters not including the ENTER character. In the example above, a valid input might result in AD$="250 N.S. MEMORY LANE, COMPUTER CITY, CA."

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# LINE INPUT#*buf#,string variable*

EXAMPLES

1000 LINE INPUT#3,AD$ *input line from disk*

DESCRIPTION

LINE INPUT# inputs a line of text from a disk file. LINE INPUT# is unlike INPUT# in that commas and other delimiters are treated as normal text characters and not as data items. The line is input from the disk file up to an ENTER character (not preceded by down arrow), the end of file, or the 255th data character. The resulting string variable includes all characters not including the ENTER character. The buf# parameter is the disk buffer associated with the file by a prior OPEN statement. LINE INPUT# can be used to input BASIC program lines when the program has been saved in ASCII format, or for other applications involving line-oriented text files.

RELATED COMMANDS

LINE INPUT

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

LIST
LIST *nnn-mmm*
LIST *-mmm*
LIST *nnn-*
line# LIST

EXAMPLES

LIST 100-999 *lists all statements from* 100 *through* 999
LIST -9000 *lists all statements from beginning through* 9000
LIST 100- *lists all statements from* 100 through end

DESCRIPTION

LIST is normally used in the command mode to list the current BASIC program in RAM to the video display. Listing will occur as rapidly as the BASIC interpreter can display the BASIC statements, and the display will "scroll" as successive statements are displayed. The program will be listed as a succession of BASIC statements in ASCII format. The display can be temporarily stopped at any time by pressing "SHIFT, @"; pressing any key will restart the listing. LIST used in the "nnn-mmm", "-mmm" or "nnn-" formats will list from a beginning line through an ending line.

RELATED COMMANDS

LLIST

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

LLIST
LLIST *nnn-mmm*
LLIST *-mmm*
LLIST *nnn-*
line# LLIST

EXAMPLES

1000 REM LLIST PROGRAM TO LINE PRINTER
3000 LLIST
LLIST 100-999 *lists all statements from* 100 *through* 999
LLIST -9000 *lists all statements from beginning through* 9000
LLIST 100- *lists all statements from* 100 *through end*

DESCRIPTION

LLIST is normally used in the command mode to list the current BASIC program in RAM to the system line printer. LLIST is logically equivalent to LIST, used for displaying the program on the video display. Only BASIC statements will be listed; no variables or other program parameters will be displayed. The program will be listed as a succession of BASIC statements in ASCII format. LLIST used in the "nnn-mmm", "-mmm" or "nnn-" formats will list from a beginning line through an ending line. LLIST alone lists the entire program.

RELATED COMMANDS

LIST

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

LOAD *"filename"*
LOAD *"filename",R*
line# LOAD *"filename"*
line# LOAD *"filename",R*

EXAMPLES

LOAD ''ACCOUNTS/BAS:1'' *load accounts payable*

DESCRIPTION

LOAD loads a BASIC program from disk. If LOAD is used without the R option, LOAD will clear all variables, close all open files and return to the BASIC command mode. If LOAD is used with the "R" option, LOAD will clear all variables, will not close open files, and will load and execute the BASIC program from its first line. LOAD in either form may be used in a BASIC statement during BASIC program execution. The "filename" is a filespec for a BASIC program stored on disk; it conforms to the general requirements for filespecs - name, extension, password, and drive number. LOAD may be used in BASIC programs to "chain" programs, allowing one program to call another in a chain of "overlays".

RELATED COMMANDS

RUN

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

LOADM*"filename"*
LOADM *"filename",offset*

EXAMPLES

LOADM ''GRAPHC'' *load file "GRAPHC" into RAM*

DESCRIPTION

LOADM is a Color Computer Disk BASIC command used to load a machine-language file from disk. The disk file must have been created by the SAVEM command. If the filename is specified without an extension, BASIC assumes that the extension is "/BIN"; this is the normal default extension for the SAVEM command. If the file is a machine-language program, an EXEC can be performed after the LOADM to execute the program; BASIC will start execution at the execution address specified in the file. If an optional offset is included, the offset constant will be added to the normal file load address, and the program or data will be "relocated" to the resulting RAM addresses. If the normal load address was &H3000 to &H30FF and the offset was &H500, for example, the data would be loaded into RAM locations &H3500 to &H35FF. Specifying an offset bias will not properly relocate machine-language code.

RELATED COMMANDS

EXEC, SAVEM

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

*line#...*LOC*(buf#)...*

EXAMPLES

1000 IF LOC(3)=5 THEN S=1 *test for fifth record*

DESCRIPTION

LOC is used to find the number of the current record in a file. The buf# parameter specifies the buffer number associated with the file. An OPEN must have been performed for the buffer (file) involved. As records are read in from the file by GET (or INPUT# for sequential files), BASIC maintains the current record number of the file and returns this number when LOC is executed. LOC is used to detect a specific record number as records are read in from disk, or in any processing that is "record dependent".

RELATED COMMANDS

LOF, OPEN

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

*line#...*LOF*(buf#)...*

EXAMPLES

`1000 FOR 1 TO LOF(3)` *loop through n records*

DESCRIPTION

LOF is used to find the number of the last record in a file. The buf# parameter specifies the buffer number associated with the file. An OPEN must have been performed for the buffer (file) involved. Once the OPEN is done, BASIC knows the number of records contained in the file and returns this number when LOF is executed. The LOF can be used to set up a processing loop for the records in the file. LOF is used as an alternative to detecting the last record number by EOF or knowing the number of records in the file beforehand.

RELATED COMMANDS

EOF, OPEN

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*LOG*(expression)...*

EXAMPLES

`1000 DB=10*(LOG(P2/P1)/LOG(10))` *find decibels*

DESCRIPTION

LOG finds the natural logarithm of a constant, variable, or expression, the logarithm to the base e, or 2.718...To find the logarithm of the argument to another base, use the formula log of X to base b=log of X to base e/log of X to base b, as in the example above. Natural logarithms are commonly used in mathematical and scientific applications.

RELATED COMMANDS

EXP

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# `LPRINT` *item list*

EXAMPLES

```
1000 LPRINT "THIS IS THE
RESULT";RS,"N=";N
```

DESCRIPTION

LPRINT is used to print a list of items on the system line printer. LPRINT is the line printer equivalent of the PRINT command. The items may be string literals (text), string variables, or numeric variables. Commas may be used between the items to tab to the next print zone, or semicolons may be used to avoid spaces between items (see "," and ";"). There may be any number of items in the list, compatible with the maximum BASIC line length. Positive numbers are printed with a leading and trailing blank. Negative numbers are printed with a minus sign and trailing blank. Strings are printed with no leading or trailing blanks. If the last item in the item list is terminated by a semicolon, the next PRINT starts from where the current PRINT left off. There are certain codes unique to various line printers which control line feeds, expanded printing, and special functions. These may be embedded in the item list by use of CHR$ or STRING$.

RELATED COMMANDS

`",", ";", PRINT`

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# `LPRINT USING` *string;item list*

EXAMPLES

`1000 A$="**$###.## DOLLARS"` *define string*

`1010 LPRINT USING A$;TOTAL` *print check*

DESCRIPTION

LPRINT USING is used for printing special formats on the system line printer, primarily dollar amounts and accounting values. The string parameter is a literal or variable string that defines the format to be used in the printing. The item list is a list of numeric or string variables that define the items to be printed. If there is more than one item, all items will be printed in the format defined by the string. The string uses "field specifiers" to define certain formats. A "#" specifies a digit position. A "." is a decimal point position and is printed in the position specified. A "," is printed in the position specified. Asterisks (*) fill unused positions left of the decimal with asterisks. "$$" or "**$" indicate a floating dollar sign, printed before the number. The string "**$###,###.## DOLLARS" used with variable A=96654.678 generates *$96,654.68 DOLLARS. Other specifiers include up arrows, plus sign, minus sign, %spaces%, and exclamation point.

RELATED COMMANDS

PRINT USING

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# LSET *field name=string*

EXAMPLES

`1000 LSET NM$=A$` *store addressee name*

DESCRIPTION

LSET is used to place character data into a random-file buffer. The normal sequence of operations establishing a random-file buffer is as follows: Define the fields of the buffer by a FIELD statement. The FIELD establishes the field names in the buffer. The RSET and LSET are then used to store character data in the fields of the buffer. The FIELD statement establishes the size for each buffer field. If the data to be stored by LSET is not as great as this field size, "filler spaces" would be filled on the right. If the field NM$ was 20 characters, the name "SPIRO SMITH" would be stored as "SPIRO SMITH   ." If data to be stored by LSET is greater than the field size, characters are truncated on the right. The data "SPIRO AGOUPOPOPODOUPOLIS" would be stored as "SPIRO AGOUPOPOPODOUP".

RELATED COMMANDS

FIELD, RSET

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*MEM...

EXAMPLES

`PRINT MEM`

`1000 PRINT MEM` *display memory left*

DESCRIPTION

MEM is a special system function that computes the amount of RAM memory currently available. The BASIC interpreter finds the amount of memory used for BASIC programs, variables, arrays, strings, stack, and reserved memory in upper RAM, subtracts it from the maximum RAM initially available and reports the result for the MEM function. This MEM value changes "dynamically" as new variables are added, string variables are computed, and so forth. MEM may be used from the command mode to find the size of a BASIC program indirectly (MEM before loading minus MEM after loading) or in a BASIC program to compare the memory currently available with memory required.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

MERGE *"filename"*
MERGE *"filename",R* *(Color Computer)*

EXAMPLES

MERGE "ACCOUNTS/BAS:1" *merge accounts payable*

DESCRIPTION

MERGE loads a BASIC program from disk and appends it to the BASIC program in RAM. The program specified in the MERGE command must be in ASCII format. (It must have been SAVEd with the "A" option.) The "filename" is a filespec for a BASIC program stored on disk; it conforms to the general requirements for filespecs - name, extension, drive number, and password. In general, the numbering of the program lines to be MERGEd from disk and the program in RAM must be mutually exclusive. If the line numbers are different, the resulting program will be made up of the line numbers from both programs in sequence. If any line numbers are the same, the lines from the disk program will replace the lines of the program in RAM. The "R" option for the Color Computer runs the program after the merge.

RELATED COMMANDS

LOAD, SAVE

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*MID$*(string,p,n)...*

EXAMPLES

1000 A$=MID$(B$,5,2) *set A$ equal to the 5th and 6th characters of B$*
1010 C$=MID$(B$,1,5) *set C$ equal to* LEFT$(B$,5)

DESCRIPTION

MID$ returns a "substring" within a larger string. The "string" parameter is the larger string to be used. The p parameter is the beginning position of the substring and may be 1 through 255. The n parameter is the length of the substring to be created and may be 1 through 255. This command takes the specified portion from the middle of the larger string and creates a new string. Suppose we have the string "MISSISSIPPI" for A$. Setting B$=MID$(A$,1,4), B$=MID$(A$,2,4), B$=MID$(A$,3,4), and B$=MID$(A$,8,4) produces B$ of "MISS", "ISSI", "SSIS", and "IPPI", respectively. If n is larger than the remaining portion of the string, the entire remainder of the string is returned. MID$ is useful for processing substrings located within larger strings for ease of handling.

RELATED COMMANDS

LEFT$, RIGHT$

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# MID$*(string,p,n)=replacement string...*

EXAMPLES

1000 MID$(A$,V,5)=''93555'' *change to new ZIP*

DESCRIPTION

MID$ normally returns a substring within a larger string. MID$= uses MID$ to find the substring and replace it with a given string or portion of a given string. The substring and replacement strings are normally the same length. The string parameter is a string variable containing the substring. The p parameter is the beginning position of the substring and may be 1 through 255. The n parameter is the length of the substring. If A$ in the above example was "COMPUTER CITY, CA 92692" and V was 19, then the substring would be "92692". The MID$ function replaces the substring with the given string if found. In this example, the new string would be "COMPUTER CITY, CA 93555". If the replacement string is greater than the length n, only n characters of the replacement string will be used. If the replacement string in the above example was "93555-1234", only the first 5 characters would be used.

RELATED COMMANDS

MID$

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*MKD$*(double-precision variable)...*

EXAMPLES

1000 A$=MKD$(A#) *convert A# to string*

DESCRIPTION

MKD$ is used to convert a double-precision numeric variable to a "string-type" variable. MKD$ is normally used to fill a random-access buffer with data values (see LSET, RSET). The typical sequence in filling a random-access buffer is to define the fields in a random-access buffer with FIELD, to convert numeric variables using MKD$, MKI$, and MKS$, to store the result with LSET and RSET and other commands, and to write out the buffer to disk. The MKD$ function converts a given double-precision variable to an 8-byte string. The 8 bytes of the string are the double-precision encoding of the numeric data and do not represent ASCII characters. They are simply a convenience in storing the data in the random-access buffer. The CVD reconverts the data to numeric form on a subsequent read. The MKD$ command can also be used to convert to a normal string variable, which is unrelated to a random buffer field name. In this case also, the string variable will be 8 bytes long.

RELATED COMMANDS

CVD, CVI, CVS, FIELD, MKI$, MKS$, LSET, RSET

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*MKI$*(integer variable)...*

EXAMPLES

```
1000 A$=MKI$(A%)
```
convert A% to string

DESCRIPTION

MKI$ is used to convert an integer numeric variable to a "string-type" variable. MKI$ is normally used to fill a random-access buffer with data values (see LSET, RSET). The typical sequence in filling a random-access buffer is to define the fields in a random-access buffer with FIELD, to convert numeric variables using MKD, MKI$, and MKS$, to store the result with LSET and RSET and other commands, and to write out the buffer to disk. The MKI$ function converts a given integer variable to a 2-byte string. The 2 bytes of the string are the integer encoding of the numeric data and do not represent ASCII characters. They are simply a convenience in storing the data in the random-access buffer. The CVI reconverts the data to numeric form on a subsequent read. The MKI$ command can also be used to convert to a normal string variable, which is unrelated to a random buffer field name. In this case also the string variable will be 2 bytes long and be made up of the numeric data of the integer variable.

RELATED COMMANDS

CVD, CVI, CVS, FIELD, MKD$, MKS$, LSET, RSET

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

*line#...*MKN$*(variable)...*

EXAMPLES

```
1000 A$=MKN$(A)
```
convert A to string

DESCRIPTION

MKN$ is used to convert a numeric variable to a "string-type" variable. MKN$ is normally used to fill a direct-access buffer with data values (see LSET, RSET). The typical sequence in filling a direct-access buffer is to define the fields in a direct-access buffer with FIELD, to convert numeric variables using MKN$, to store the result with LSET and RSET and other commands, and to write out the buffer to disk. The MKN$ function converts a given variable to a 5-byte string. The 5 bytes of the string are the binary encoding of the numeric data and do not represent ASCII characters. They are simply a convenience in storing the data in the direct-access buffer. The CVN reconverts the data to numeric form on a subsequent read. The MKN$ command can also be used to convert to a normal string variable, which is unrelated to a buffer field name. In this case also, the string variable will be 5 bytes long and be made up of the numeric data of the numeric variable.

RELATED COMMANDS

CVN, FIELD, RSET, LSET

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*MKS$*(single-precision variable)...*

EXAMPLES

```
1000 A$=MKS$(A)
```
convert A to string

DESCRIPTION

MKS$ is used to convert a single-precision numeric variable to a "string-type" variable. MKS$ is normally used to fill a random-access buffer with data values (see LSET, RSET). The typical sequence in filling a random-access buffer is to define the fields in a random-access buffer with FIELD, to convert numeric variables using MKD$, MKI$, and MKS$, to store the result with LSET and RSET and other commands, and to write out the buffer to disk. The MKS$ function converts a given single-precision variable to an 4-byte string. The 4 bytes of the string are the double-precision encoding of the numeric data and do not represent ASCII characters. They are simply a convenience in storing the data in the random-access buffer. The CVS reconverts the data to numeric form on a subsequent read. The MKS$ command can also be used to convert to a normal string variable, which is unrelated to a random buffer field name. In this case also the string variable will be 4 bytes long.

RELATED COMMANDS

CVD, CVI, CVS, FIELD, MKD$, MKI$, LSET, RSET

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...expression MOD *expression...*

EXAMPLES

```
1000 C=A MOD B
```

DESCRIPTION

MOD is a numeric operator that performs a "modulus" arithmetic operation on two operands and returns a result. The two operands involved (constants, variables, or expressions) are converted to two-integer operands. A modulus operation divides the first operand by the second operand and finds the remainder. The remainder is then returned as the result of the modulus operation. If the first operand is 100, and the second is 44, the result of 100 MOD 44 is the remainder of 100/44, or 12. Modulus arithmetic is useful in such processing as finding the "12-" or "24-hour clock" times (elapsed hours MOD 12 or 24).

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

MOTOR ON
line# MOTOR ON
MOTOR OFF
line# MOTOR OFF

EXAMPLES

```
1000 MOTOR ON
3000 MOTOR OFF
```

DESCRIPTION

MOTOR ON turns on the cassette motor by activating the cassette "remote" output. The motor will remain on until a MOTOR OFF command is executed. MOTOR ON can be used to automatically control the cassette motor for positioning or other uses from within a BASIC program. (The motor is automatically turned on, however, by the CLOAD and CLOADM commands.) MOTOR OFF deactivates the remote output and turns the cassette motor OFF.

RELATED COMMANDS

CLOAD, CLOADM

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

NAME *newline,startline,increment*

EXAMPLES

NAME 100,300,5 *from line* 100 *with start of* 300*, increment of 5*

DESCRIPTION

NAME renumbers the current BASIC program in RAM. All line numbers in the program will be changed to a new range of numbers, starting with a given number, and with a given increment. This includes not only statement line numbers at the beginning of BASIC lines, but line numbers referenced by GOTOs, GOSUBs, THENs, ON...GOTOs, and ON...GOSUBs. The newline parameter is the starting line number of the program after renumbering. The startline parameter is the first line number of the current program from which renumbering is to occur. The increment parameter is the increment to be used between new line numbers. All parameters are optional. Defaults are 10 for "newline", 10 for "increment", and the entire program for "startline". Commas can be used for missing parameters, or NAME can be used by itself without parameters to renumber the entire program with new line numbers from 10 in increments of 10.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

NEW
line# NEW

EXAMPLES

```
NEW      erase old BASIC program
```

DESCRIPTION

NEW clears any current BASIC program in RAM, resets all variables to 0, and generally reinitializes all BASIC parameters. It does not affect non-BASIC data, such as reserved memory areas for machine-language programs. NEW should be used to "erase" the current BASIC program in memory in preparation for entering a new program from the keyboard. NEW does not have to be used prior to loading in a new BASIC program from disk or cassette. NEW would not normally be used in a BASIC program statement, as it produces catastrophic results and destroys the program.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# NEXT *variable*
line# NEXT

EXAMPLES

```
1000 FOR I=1 TO 100 loop 100 times
1010 PRINT I  print variable
1020 NEXT I  loop
```

DESCRIPTION

The NEXT command is used together with FOR...TO...NEXT to set up and execute a program loop. The FOR...TO...NEXT statement defines the start, end, and increment values for a variable "counter" used to determine the number of passes through the loop. Any number of statements may be placed between the FOR...TO...STEP and NEXT statements. The variable in NEXT is optional. Any number of FOR...TO...STEP loops may be "nested". In this case, the innermost NEXT must always use the variable associated with the innermost FOR...TO...NEXT statement. The NEXT statement increments the loop variable by the STEP size, and if the variable has not exceeded the end value, control is returned back to the FOR...TO...STEP statement. The loop may be broken with a GOTO or similar transfer at any time. The variable controlling the loop may also be altered in statements other than the NEXT.

RELATED COMMANDS

FOR...TO...STEP

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*NOT*(expression)...*

EXAMPLES

```
1000 IF NOT (A<B) THEN PRINT
""HELP!""
1010 A=NOT(B-1) two's complement
```

DESCRIPTION

NOT is used as a relational operator and for bit manipulation. In the first use, NOT tests a constant, variable, or expression. If the expression is false, then the NOT function is true. In the example above, NOT (A<B) is true if variable A is greater or equal to variable B. The THEN action would not be taken if A was less than B. In the bit manipulation case, NOT is used to perform a one's complement on an integer variable or end product of an expression. A one's complement operates on binary values. It "inverts" each bit, changing a one to a zero and a zero to a one. The NOT in this application can be used to invert bits and perform other bit-wise operations.

RELATED COMMANDS

AND, OR

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line#...*OCT$*(expression)...*

EXAMPLES

```
1000 PRINT OCT$(A)    find octal value of A
```

DESCRIPTION

OCT$ is a special function that will convert a constant, variable, or expression to a string that represents the octal value of the argument. OCT$(1000), for example, will be converted to the string "1750". Octal notation is used primarily for machine-language operations in specifying addresses, instruction codes, and data values.

RELATED COMMANDS

&O

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ON ERROR GOTO *line#*
line# ON ERROR GOTO 0

EXAMPLES

1000 ON ERROR GOTO 10000 *define error-processing routine*

DESCRIPTION

ON ERROR GOTO is used to define the line number of a user error-processing routine. ON ERROR GOTO should be defined early in the program before errors can occur. After ON ERROR GOTO is executed with a valid line number, the user error-processing mode is in force, and all errors that occur will cause a transfer to the line number of the error-processing routine. The user error-processing routine can be disabled by executing an ON ERROR GOTO 0 command. Disabling user error-processing will return to the BASIC interpreter's normal error action. The error-processing routine normally contains code that will detect the type of error (see ERR) and the line in which the error occurred (see ERL), in addition to code to report the error to the user and recommend corrective action. In some cases, the normal BASIC error action will be reinstated (see RESUME).

RELATED COMMANDS

ERL, ERR, ERROR, RESUME

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# ON *expression* GOSUB *line# 1, line# 2,...,line# n*

EXAMPLES

1000 ON AX GOSUB 100,200,300,400,500 *does a* GOSUB *to* 100 *if AX=1,* 200 *if AX=2,...*

2000 ON (B-5) GOSUB 1000,2000,3000,234 *does a* GOSUB *to* 1000 *if (B-5)=1,* 2000 *if (B-5)=2,...*

DESCRIPTION

This is a "computed GOSUB". The quantity before the GOSUB may be a constant (trivial), variable, or expression. The integer portion of the quantity is found. If this is 1, 2, 3, etc., the first, second, third, etc. line number is found and a GOSUB to the line number performed. If the integer portion is 0, or greater than the number of line numbers, the next statement in sequence is executed. If the integer portion is negative or greater than 255, an error occurs. The computed GOSUB allows "branching out" to a number of subroutines based on a single variable:

```
1000 REM BRANCH OUT ON MENU
SELECTION 1-5
1010 ON N GOSUB
1000,2000,3000,4000,5000
1020 REM NOT 1-5 HERE OR RETURN
POINT
```

RELATED COMMANDS

GOSUB, ON...GOTO

SYSTEM	
I, LVL I	●
I, LVL II	●
I, Disk	●
II	●
III, LVL I	●
III, LVL III	●
III, Disk	●
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

line# ON *expression* GOTO *line# 1, line# 2,...,line# n*

EXAMPLES

1000 ON AX GOTO 100,200,300,400,500
does a GOTO *to* 100 *if AX=1,* 200 *if AX=2,...*
2000 ON (B-5) GOTO
1000,2000,3000,234 *does a* GOTO *to* 1000 *if (B-5)=1,* 2000 *if (B-5)=2,...*

DESCRIPTION

This is a "computed GOTO". The quantity before the GOSUB may be a constant (trivial), variable, or expression. The integer portion of the quantity is found. If this is 1, 2, 3, etc., the first, second, third, etc. line number is found and a GOTO to the line number performed. If the integer portion is 0, or greater than the number of line numbers, the next statement in sequence is executed. If the integer portion is negative or greater than 255, an error occurs. Normally the quantity would be a single variable or expression. The computed GOTO allows "branching out" to a number of lines based on a single variable, such as a menu selection:

```
1000 REM BRANCH OUT ON MENU
SELECTION 1-5
1010 ON N GOTO
1000,2000,3000,4000,5000
1020 REM NOT 1-5 HERE
```

RELATED COMMANDS

GOTO, ON...GOSUB

SYSTEM	
I, LVL I	
I, LVL II	
I, Disk	●
II	●
III, LVL I	
III, LVL III	
III, Disk	●
CC, BASIC	
CC, Ext BASIC	
CC, Disk	●

FORMAT

line# OPEN *mode,buf#,filename*
line# OPEN *mode,buf#,filename,rec-length*

EXAMPLES

1000 OPEN ''O'',1,''PAYABLE:1''
open payables file

DESCRIPTION

OPEN causes BASIC to initiate, extend, or locate a disk file, to establish a RAM buffer for disk operations, and to establish a record length. The mode parameter is a one-character string that establishes the basic operation. "I" specifies sequential input starting at the first record. "O" specifies sequential output starting at the first record. If the filename does not exist, a new file is created. "E" (not used in the Color Computer.) appends output to the end of an existing file (or creates a new file). "R" ("D" in the Color Computer for "direct-access" file) specifies random input/output of a file. If mode is a constant, it must be enclosed in quotes. The buf# parameter is a numeric value specifying the buffer number. The filename parameter is a standard file specification. A constant must be enclosed in quotes. The rec-length parameter is optional for the "R" mode. If not used, 256 bytes is used for the length.

RELATED COMMANDS

CLOSE

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...(expression) OR *(expression)...*

EXAMPLES

```
1000 IF (A<2) OR (B>5) THEN PRINT
''HELP!''
1010 A=A OR B set bit 3
```

DESCRIPTION

OR is used as a relational operator and for bit manipulation. In the first use, OR compares two constants, variables, or expressions. If either expression is true, then the OR function is true. In the example above, (A<2) AND (B>5) is true if variable A is less than 2 OR variable B is greater than 5. The THEN action would only be taken if either expression was true (expression 1 OR expression 2). In the bit manipulation case, OR is used to logically OR integer variable bits, considered to be binary numbers. An OR of binary values produces a 1 for each bit position if either operand has a 1 bit in that bit position. An OR of the two binary values 10100000 and 11001111 would produce a result of 11101111. The OR in this application can be used to test bits, set individual bits, and perform other bit-wise operations.

RELATED COMMANDS

AND, NOT

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# OUT *port,value*

EXAMPLES

```
1000 OUT 255,2   turn cassette on
1010 OUT 255,1   turn cassette off
1030 GOTO 1000   loop
```

DESCRIPTION

OUT is a command that outputs a one-byte value to a system I/O port. The Model I/II/III use I/O ports for certain system devices such as cassette or RS-232-C. The OUT enables a BASIC program to directly output data to these I/O ports. The port parameter is an address value of 0 through 255 that defines the I/O address. The value parameter is a one-byte value of 0 through 255 that represents the data to be output to the I/O port.

RELATED COMMANDS

INP

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PAINT*(x,y),c,b*

EXAMPLES

1000 PAINT (120,100),3,4 *paint with blue until red*

DESCRIPTION

The PAINT command colors an area on a graphics screen. The x,y coordinate defines a starting point for the paint. The x,y coordinates are in "high-resolution" coordinates of 0-255 and 0-191. The c and b parameters are standard color code of 1 through 8 (green, yellow, blue, red, buff, cyan, magenta, and orange). The c parameter defines the color for the paint; the b parameter defines the "boundary" color. The painting will "spread out" from the starting point until the specified boundary color is encountered . If the boundary color is not found, or if it does not completely contain the PAINT area, the PAINT operation will continue over the entire screen (or until a proper boundary condition).

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PCLEAR *n*

EXAMPLES

line# PCLEAR 8 *clear 8 graphics pages*

DESCRIPTION

PCLEAR reserves n number of graphics pages. The graphics pages are separate from the text screen in the Color Computer. Each graphics page is 1536 bytes long, and up to 8 pages may be used for display of graphics data. Depending upon the PMODE in force, anywhere from 1 to 4 pages may be on display at any time; the remaining pages are used as storage for additional graphics data. The starting page number may be changed by the PMODE command. If PCLEAR is never executed, the default number of graphics pages reserved is 4. PCLEAR does not clear the graphics pages (see PCLS).

RELATED COMMANDS

PCLS, PMODE, SCREEN

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC •
CC, Disk •

FORMAT

line# PCLS *color*

EXAMPLES

1000 PCLS 8 *clear the screen to orange*

DESCRIPTION

PCLS is the Extended Color BASIC equivalent of the CLS command. It clears the current graphics screen with the specified color. Valid colors are 1 through 8, representing green, yellow, blue, red, buff, cyan, magenta, and orange, respectively. The color specified must be in the color set currently selected. If the color selected is not in the current color set, the screen will be cleared to a "corresponding" color in the current color set. PCLS 8 while in color set 0, for example, will clear the graphics display to red if in a four-color mode. PCLS 8 while in color set 0 and a two-color mode will clear the graphics screen to black. The graphics screen does not have to be on display for the PCLS to take effect. As the graphics pages are separate from the text screen, they can be cleared independently.

RELATED COMMANDS

SCREEN

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC •
CC, Disk •

FORMAT

line# PCOPY *n* TO *m*

EXAMPLES

1000 PCOPY 1 TO 8

DESCRIPTION

PCOPY is used to copy the contents of one graphic page to another graphics page. There are 8 graphics pages in Extended Color BASIC in the Color Computer, numbered 1 through 8. Any page may be copied to another page for purposes of initialization or temporary storage. PCOPY copies only the 1536 bytes of one page (n) to another (m). If the graphics mode in force uses more than one page for graphics display, then more than one PCOPY may have to be done to display all of the graphics data. The "source" page, the page to be copied, remains unaltered after the copy.

RELATED COMMANDS

PMODE

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*PEEK*(expression)...*

EXAMPLES

```
1000 FOR I=31000 TO 31000+14 set up loop
1010 PRINT PEEK(I) print byte
1020 NEXT I continue
```

DESCRIPTION

PEEK is a function that allows you to look at a byte of memory in ROM, RAM, or "memory-mapped" I/O device. It returns the contents of a single memory location whose address is specified by a constant, variable, or expression within parentheses after the PEEK. As all memory locations in the TRS-80 systems contain 8 bits or one byte of data, the contents will be a value from 0 through 255. PEEK can be used in conjunction with POKE to process bytes of memory for combining BASIC programs with machine-language programs. PEEK can also be used to examine certain I/O devices whose addresses simulate memory locations.

RELATED COMMANDS

POKE

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PLAY *string*

EXAMPLES

```
1000 PLAY ''C;D;E;F;G;A;B;C''  play scale
```

DESCRIPTION

PLAY plays a string of musical notes with control of frequency, note length, tempo, volume, and pauses. The "string" argument is a string constant or variable that defines the PLAY operations. The general format is a series of "subcommands" separated by semicolons. The letters from A through G specify note value subcommands. A suffix of "+", "#" indicates a sharp, and "–" indicates a flat. (A# is A sharp.) N1 through N12 also indicate note values. O followed by 1 through 5 indicate the octave. L followed by 1 through 255 indicates the note length (1 is a whole note, 2 a half note, 4 a quarter note, etc.) T followed by 1 through 255 is tempo, slow to fast. V followed by 1 through 31 is volume, low to high. P followed by 1 through 255 is pause length. Substrings may be executed by X followed by substring to be executed.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PMODE *mode,start-page*

EXAMPLES

1000 PMODE 3,1 *select* PMODE 3, *start-page 1*

DESCRIPTION

PMODE is used to select the graphics resolution and starting graphics page number in Extended Color Basic. The mode parameter selects one of 5 modes, numbered 0 through 4. The resolution of the graphics screen increases with the mode number. Mode 0 is a two-color 128 by 96 mode, mode 1 is a four-color 128 by 96 mode, mode 2 is a two-color 128 by 192 mode, mode 3 is a four-color 128 by 192 mode, and mode 4 is a two-color 256 by 192 mode. The color set displayed depends upon the SCREEN command. Two-color modes display black on green (set 0) or black on buff (set 1). Four color modes display green, yellow, blue, red (set 0) or buff, cyan, magenta, orange (set 1). The start-page may be any graphics page from 1 to 8. The PMODE command does not cause a display of the graphics page; SCREEN sets either a text display or graphics data.

RELATED COMMANDS

SCREEN

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# POINT*(X,Y)*

EXAMPLES

1010 A=POINT(63,31) *read contents of pixel*

DESCRIPTION

Model I/III: POINT is used to test one graphics "pixel". There are 6144 pixels, divided up as 128 horizontal elements by 48 vertical elements. The POINT command tests one of these pixels for "on" or "off" status. Each of the 6144 pixels can be uniquely tested. The x coordinate specifies the horizontal position of 0-127. The y coordinate specifies the vertical position of 0-47. If the point is "on", POINT returns a -1. If the point is "off", POINT returns a 0.

Color Computer: POINT is used to test one graphics "pixel" for "off" or "on". There are 2048 pixels, divided up into 64 horizontal elements by 32 vertical elements. The x coordinate specifies the horizontal position of 0-63. The y coordinate specifies the vertical position of 0-31. If the point is "off", a 0 is returned. If "on" in the graphics mode, the color code of 1 through 8 (green, yellow, blue, red, buff, cyan, magenta, orange) is returned. If in the character mode, a -1 is returned.

RELATED COMMANDS

CLS, RESET, SET

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*POKE *expression,value...*

EXAMPLES

```
1000 FOR I=31000 TO 31000+14 set up loop
1010 POKE I,0 clear bytes
1020 NEXT I continue
```

DESCRIPTION

POKE is a function that allows you to store data in memory locations in RAM, or "memory-mapped" I/O devices. A value of 0 through 255 is stored in the memory location specified by a constant, variable, or expression. As all memory locations in the TRS-80 systems contain 8 bits or a byte of data, values greater than 255 are not valid. POKE can be used in conjunction with PEEK to process bytes of memory for combining BASIC programs with machine-language programs. POKE can also be used to output to certain I/O devices whose addresses simulate memory locations.

RELATED COMMANDS

PEEK

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*POS*(dummy)...*

EXAMPLES

```
1000 PRINT V;TAB(POS(0)+3) insert 3 spaces
```

DESCRIPTION

POS is a function that returns the current cursor position of the video display, from 0 through 63 (Model I/III), 0 though 79 (Model II), or 0 through 31 (Color Computer). POS may be used for columnization or word-processing applications.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC •
CC, Disk •

FORMAT

line# PPOINT*(x,y)*

EXAMPLES

```
1000 PPOINT (128,96)  test middle element
```

DESCRIPTION

PPOINT is used to test one graphic element on the current graphics page. The "x" and "y" parameters define the horizontal and vertical element numbers, respectively. The x value can range from 0 through 255; the y value can range from 0 through 191. The coordinates specify an element in the highest graphics resolution of 256 by 192 elements. The actual area tested depends upon the current PMODE resolution for graphics. The element will be tested even if the current display is of the text page. PPOINT returns the color code for the graphics element defined by x and y. Color codes are 1 through 8 defining colors of green, yellow, blue, red, buff, cyan, magenta, and orange.

RELATED COMMANDS

PRESET, PSET

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC •
CC, Disk •

FORMAT

line# PRESET *(x,y)*

EXAMPLES

```
1000 PRESET (129,96)  reset middle dot
```

DESCRIPTION

PRESET is used to reset one graphic element on the current graphics page. The x and y parameters define the horizontal and vertical element numbers, respectively. The x value can range from 0 through 255; the y value can range from 0 through 191. The coordinates specify an element in the highest graphics resolution of 256 by 192 elements. The actual area reset depends upon the current PMODE resolution set for graphics. The element will be reset regardless of the display of the current page. The color used for the reset action is the current background color. If SCREEN has specified the text page, no action will be seen, but the PRESET action has occurred. "PRESET" is also used in the LINE command, where it means "draw the line or box in current background color", effectively "resetting" the line.

RELATED COMMANDS

LINE, PSET

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PRINT *item list*

EXAMPLES

```
1000 PRINT ``THIS IS THE
RESULT '';RS,``N='';N
```

DESCRIPTION

PRINT is used to display a list of items on the video display. The items are generally printed on one line or a portion of one line. The items may be string literals (text), string variables, or numeric variables. Commas may be used between the items to tab to the next print zone, or semicolons may be used to avoid spaces between items (see "," and ";"). The example above prints one line of "THIS IS THE RESULT XXX   N= XXX", where XXX represents the value of variables RS and N. There may be any number of items in the list, compatible with the maximum BASIC line length. Positive numbers are printed with a leading and trailing blank. Negative numbers are printed with a minus sign and trailing blank. Strings are printed with no leading or trailing blanks. If the last item in the item list is terminated by a semicolon, the next PRINT starts from the point at which the current PRINT left off.

RELATED COMMANDS

``,'', ``;''

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PRINT#-1,*item list*

EXAMPLES

```
1000 PRINT#-1,A,B,C$,``****''
```

DESCRIPTION

PRINT#-1 outputs the specified item list to cassette tape. The cassette tape must have been positioned to the proper point for file output. PRINT#-1 is similar to the PRINT display statement. It outputs character strings to the cassette after converting numeric variables. Any number of items may be used in the item list in any combination of constants, numeric variables, string literals, or string variables. Each item must be separated by a delimiter of a comma or semicolon. The maximum length of characters output to tape must not exceed 248; this is a function of the number and lengths of items in the list. Items output to a cassette file can be read in by the INPUT#-1 command; input must be in the same sequence as output.

RELATED COMMANDS

INPUT#-1,PRINT

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

line# PRINT#-2, *item list*

EXAMPLES

```
1000 PRINT#-2, ''THIS IS THE
RESULT'';RS,''N='';N
```

DESCRIPTION

PRINT#-2 is used to print a list of items on the system line printer. PRINT#-2 is the line printer equivalent of the video display PRINT command. The items may be string literals (text), string variables, or numeric variables. Commas may be used between the items to tab to the next print zone, or semicolons may be used to avoid spaces between items (see "," and ";"). There may be any number of items in the list, compatible with the maximum BASIC line length. Positive numbers are printed with a leading and trailing blank. Negative numbers are printed with a minus sign and trailing blank. Strings are printed with no leading or trailing blanks. If the last item in the item list is terminated by a semicolon, the next PRINT starts from the last PRINT position. There are certain codes unique to various line printers which control line feeds, expanded printing, and special functions. These may be embedded in the item list by use of CHR$ or STRING$.

RELATED COMMANDS

'','', '';'', PRINT

SYSTEM

I, LVL I	
I, LVL II	●
I, Disk	●
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# PRINT#-2,*item list*

EXAMPLES

```
1000 PRINT#-2,A,B,C$,''****''
```

DESCRIPTION

PRINT#-2 is identical to PRINT#-1 except that it is used for the second cassette of the system. PRINT#-2 outputs the specified item list to cassette tape. The cassette tape must have been positioned to the proper point for file output. PRINT#-2 is similar to the PRINT display statement. It outputs character strings to the cassette after converting numeric variables. Any number of items may be used in the item list in any combination of constants, numeric variables, string literals, or string variables. Each item must be separated by a delimiter of a comma or semicolon. The maximum length of characters output to tape must not exceed 248; this is a function of the number and lengths of items in the list. Items output to a cassette file can be read in by the INPUT#-2 command; input must be in the same sequence as output.

RELATED COMMANDS

INPUT#-2,PRINT

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# `PRINT#`*buf#,item list*
line# `PRINT#` *buf#,*`USING` *string,item list*

EXAMPLES

```
1000 PRINT#3,A;B;C$
```
output to file

DESCRIPTION

PRINT# performs a write to a sequential disk file. The file must have been previously OPENed. The OPEN command specifies a buffer for the file name, and this buffer number is used in the PRINT# command. PRINT# outputs a list of items to the buffer (file). The items may be any number of numeric or string variables. All items are transformed into character strings and written to the disk buffer. The PRINT# output to the file is similar to the display output of PRINT. If commas are used to separate the items, spaces for tabs will be written. If semicolons are used, no spaces will be used between items. String variables should use CHR$(34) to bracket the variables with double quotes if the string variables contain delimiters such as commas or semicolons; otherwise string variables can be used in the list as required. The USING option outputs the list in the format specified by the USING string. The format is identical to that used in PRINT USING.

RELATED COMMANDS

PRINT USING

SYSTEM

I, LVL I	•
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# `PRINT#,`*item list*

EXAMPLES

```
1000 PRINT#,A,B,C$,"*****"
```

DESCRIPTION

PRINT# outputs the specified item list to cassette tape. The cassette tape must have been positioned to the proper point for file output. PRINT# is similar to the PRINT display statement. It outputs character strings to the cassette after converting numeric variables. Any number of items may be used in the item list in any combination of constants, numeric variables, string literals, or string variables. Each item must be separated by a delimiter of a comma or semicolon. The maximum length of characters output to tape must not exceed 248; this is a function of the number and lengths of items in the list. Items output to a cassette file can be read in by the INPUT# command; input must be in the same sequence as output.

RELATED COMMANDS

INPUT#, PRINT

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PRINT USING *string;item list*

EXAMPLES

1000 A$=‘‘**$###.## DOLLARS’’ *define string*

1010 PRINT USING A$: TOTAL *print check*

DESCRIPTION

PRINT USING is used for displaying special formats, primarily dollar amounts and accounting values. The string parameter is a literal or variable string that defines the format to be used in the display. The item list is a list of numeric or string variables that define the items to be printed. If there is more than one item, all items will be printed in the format defined by the string. The string uses "field specifiers" to define certain formats. A "#" specifies a digit position. A "." is a decimal point position and is printed in the position specified. A "," is printed in the position specified. Asterisks (*) fill unused positions left of the decimal with asterisks. "$$" or "**$" indicate a floating dollar sign, printed before the number. The string "**$###,###.## DOLLARS" used with variable A=96654.678 generates *$96,654.68 DOLLARS. Other specifiers include up arrows, plus sign, minus sign, %spaces%, and exclamation point.

RELATED COMMANDS

LPRINT USING

SYSTEM

I, LVL I	•
I, LVL II	
I, Disk	
II	
III, LVL I	•
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# PRINT AT *position,item list*

EXAMPLES

1000 PRINT AT 128, ‘‘THIS IS THE RESULT ’’:RS,‘‘N=’’:N

DESCRIPTION

PRINT AT is used to display a list of items on the video display at a specified starting location. The items are generally printed on one line or a portion of one line. The items may be string literals (text), string variables, or numeric variables. Commas may be used between the items to tab to the next print zone, or semicolons may be used to avoid spaces between items (see "," and ";"). In the example above, the message is printed beginning at print position 128. The Model I has 1024 print positions; each line starts with a multiple of 64. There may be any number of items in the list, compatible with the maximum BASIC line length. Positive numbers are printed with a leading and trailing blank. Negative numbers are printed with a minus sign and trailing blank. Strings are printed with no leading or trailing blanks. If the last item in the item list is terminated by a semicolon, the next PRINT starts from the point at which the current PRINT left off.

RELATED COMMANDS

‘‘,’’, ‘‘;’’, PRINT

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PRINT *@position,item list*

EXAMPLES

```
1000 PRINT @128, ''THIS IS THE
RESULT '';RS,''N='';N
```

DESCRIPTION

PRINT @ is used to display a list of items on the video display at a specified starting location. The items may be string literals (text), string variables, or numeric variables. Commas may be used between the items to tab to the next print zone, or semicolons may be used to avoid spaces between items (see "," and ";"). The Model I and III have 1024 print positions; each line starts with a multiple of 64. The Model II has 1920 print positions; each line starts with a multiple of 80. The Color Computer has 512 print position; each line starts with a multiple of 32. Print positions are numbered starting from 0. There may be any number of items in the list, compatible with the maximum BASIC line length. Positive numbers are printed with a leading and trailing blank. Negative numbers are printed with a minus sign and trailing blank. Strings are printed with no leading or trailing blanks.

RELATED COMMANDS

'',''. '';''. PRINT

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PSET *(x,y,c)*
line# PSET *(x,y)*

EXAMPLES

1000 PSET (129,96,3) *set middle dot to blue*

DESCRIPTION

PSET is used to set one graphic element on the current graphics page. The x and y parameters define the horizontal and vertical element numbers, respectively. The x value can range from 0 through 255; the y value can range from 0 through 191. The coordinates specify an element in the highest graphics resolution of 256 by 192 elements. The actual area set depends upon the current PMODE resolution set for graphics. The color parameter, c, may be any valid color number of 1 through 8 (green, yellow, blue, red, buff, cyan, magenta, and orange). Again, valid color codes depend upon the PMODE mode. The c parameter is optional; if c is omitted, the current foreground color is used. If SCREEN has specified a text page, no action will be seen, but the PSET action has occurred. "PSET" is also used in the LINE command, where it means "draw the line or box in current foreground color".

RELATED COMMANDS

LINE, PRESET

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# PUT *buf#*
line# PUT *buf#,rec#*

EXAMPLES

1000 PUT 3,100 *output 100th record*

DESCRIPTION

PUT is used to output a random-access file record to disk. A random-access file allows records to be read or written on a random basis (not in sequence). The PUT outputs the contents of the current record as the next record in sequence or as the specified record number of the random file. The "current record" is the entire buffer contents if the record length defined by the OPEN was 256, or a portion of the buffer if the record length was less than 256. Prior to the PUT, an OPEN with the "R" option must have been executed. The OPEN defines the filename and buffer associated with the file, and the file length. The PUT buf# form of PUT outputs the current record in the buffer as a record whose number is one higher than the last access. If no record has been written, this becomes the first record of the file. The second form of PUT writes the current record as the specified record number defined by "rec#".

RELATED COMMANDS

GET

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# PUT*(x1,y1)-(x2,y2),array name,action*

EXAMPLES

```
1000 PUT
(205,141)-(255,191),AA,PSET
```

DESCRIPTION

GET stores any rectangular area on a graphics screen in a two-dimensional array. A PUT later retrieves the graphics data from the array and displays it in any other area of the graphics screen. GET/PUT can be used to save portions of a graphics screen or to create animation effects. The x1,y1 coordinates define one corner of the screen area for the PUT operation; The x2,y2 coordinates define the opposing corner. The x1,x2 and y1,y2 values are in "high-resolution" graphics coordinates of 0-255 and 0-191, respectively. The "array name" is the name of a two-dimensional array previously filled by a GET statement. In general, the PUT area must be equal to the dimensions of the GET area. The "action" option is PSET, PRESET, AND, OR, or NOT. If a "G" option was used in the GET, then an action item must be used in the PUT. PSET transfer the data in the same way, PRESET inverts the colors, and AND, OR, and NOT can be used to perform logical operations on the graphics data.

RELATED COMMANDS

GET

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# RANDOM

EXAMPLES

1000 RANDOM *"reseeds" the random number generator for* RND

1010 PRINT RND(100): GOTO 1010 *print list of random numbers from 1 to 100*

DESCRIPTION

RANDOM initializes the random number generator for the RND function. The RND function is used to generate pseudo-random numbers from 0 to N. Pseudo-random numbers are "repeatable" numbers, that is, the same sequence of numbers is repeated from the same starting number. If RANDOM is never used, the same sequence of numbers will be generated on system power up or restart. The sequence will be quite long, but RANDOM ensures that a true random starting point is used for an unpredictable sequence of numbers.

RELATED COMMANDS

RND

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# READ *variable 1, variable 2, variable 3,...variable N*

EXAMPLES

1000 READ A,B,XY *reads three numeric values*
1010 READ Z%,XX% *reads two integer values*
1020 READ A$,B$ *reads two strings*

DESCRIPTION

READ reads a value or values from a DATA list. The variables in the READ are set to the next values in the DATA list. The variable types in the DATA list must correspond to the variable types in the READ statement. Variable types in the READ statement may be intermixed as long as they appear that way in the DATA list. The following statements read 5, 13, ORANGE into variables X, Y, and XY$, and then read -37, 2, and BANANA into variables A, B, and B$.

1000 DATA 5,13,ORANGE,-37,2,BANANA *establishes list*
1010 READ X,Y,XY$ *reads first three values*
1020 READ A,B,B$ *reads next three values*

RELATED COMMANDS

DATA, RESTORE

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# REM

EXAMPLES

```
1000 REM THIS PROGRAM SEGMENT IS A
SORT
1010 REM IT SORTS TWO-D ARRAY ZZ
```

DESCRIPTION

REM is an abbreviation for "remark". The REM command may be followed by descriptive text defining the program statements. REMarks "text" is not executed, but does take up BASIC program space. As many REMs as required may be used. Delete the REM statements in the final program version to save program space and increase program execution speed.

RELATED COMMANDS

•

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

RENAME *"old file"* TO *"new file"*

EXAMPLES

```
RENAME ''ACCTS/PAY:0'' TO
''ACCTS/REC:0''
```

DESCRIPTION

RENAME is a Color Computer Disk BASIC command that changes the name of a file. The "old file" and "new file" parameters are valid file names; both require extensions. File names are in the name/extension:drive# format. The drive# is optional. RENAME is normally used to rename a file on the same disk.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

RENUM *newline,startline,increment (all arguments optional)*

EXAMPLES

RENUM 100,300,5 *from line 100 with start of 300, increment of 5*

DESCRIPTION

RENUM renumbers the current BASIC program in RAM. All line numbers in the program will be changed to a new range of numbers, starting with a given number, and with a given increment. This includes not only statement line numbers at the beginning of BASIC lines, but line numbers referenced by GOTOs, GOSUBs, THENs, ON...GOTOs, and ON...GOSUBs. The newline parameter is the starting line number of the program after renumbering. The startline parameter is the first line number of the current program from which renumbering is to occur. The increment parameter is the increment to be used between new line numbers. All parameters are optional. Defaults are 10 for newline, 10 for increment, and the entire program for startline. Commas can be used for missing parameters, or RENUM can be used alone.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# RESET*(x,y)*

EXAMPLES

1010 RESET(0,0) *reset upper left-hand pixel*

DESCRIPTION

Model I/III: RESET is used to reset one graphics "pixel" to black. There are 6144 pixels, divided up as 128 horizontal elements by 48 vertical elements. The RESET command resets one of these pixels to "off". Each of the 6144 pixels can be uniquely RESET. The x coordinate specifies the horizontal position of 0-127. The y coordinate specifies the vertical position of 0-47.

Color Computer: RESET is used to reset one graphics "pixel". There are 2048 pixels, divided up into 64 horizontal elements by 32 vertical elements. The x coordinate specifies the horizontal position of 0-63. The y coordinate specifies the vertical position of 0-31. The reset turns off the pixel to a black color.

RELATED COMMANDS

CLS, POINT, SET

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# RESTORE

EXAMPLES

1000 RESTORE *resets the pointer to the* DATA *list*

DESCRIPTION

RESTORE resets the internal DATA list pointer to the beginning of the DATA list. All DATA statements scattered throughout a BASIC program (or appearing consecutively) create one contiguous list of DATA values. RESTORE resets the internal DATA list pointer to the first entry in the list so that the next READ results in a read of that entry. The following statements read 5, -27.5, and 3 into variables A, B, C and then into variables D, E, and F.

1000 DATA 5.-27.5.3.5.2.13 *establishes list*
1010 READ A.B.C *read first three values*
1020 RESTORE *resets pointer*
1030 READ D.E.F *reads first three values*

RELATED COMMANDS

DATA.READ

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# RESUME
line# RESUME 0
line# RESUME *line#*
line# RESUME NEXT

EXAMPLES

1000 RESUME NEXT *resume after error*

DESCRIPTION

RESUME is the last executed statement of a user error-processing routine. A user error-processing routine is defined by a ON ERROR GOTO command. The error-processing is entered every time an error occurs so that the program may investigate the type of error. RESUME is used after investigation of the type of error, line number, messages, and corrective action, if any. RESUME without a line number or with a line number of 0 causes the BASIC interpreter to return to the line in which the error occurred. This mode would be used after the normal BASIC error action was reinstated by an ON ERROR GOTO 0. RESUME with a line number causes a branch to the specified line number; it is a way of taking further action related to the occurrence of the error. RESUME NEXT causes a continuation of the program after the line in which the error occurred.

RELATED COMMANDS

ERL, ERR, ERROR, ON ERROR GOTO

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# RETURN

EXAMPLES

1000 GOSUB 12000 *calls subroutine at* 12000
1010 *(return point) return point from* 12090
...
12000 *(subroutine: from 1 to many statements)*
12090 RETURN *returns to statement after* GOSUB

DESCRIPTION

RETURN defines the last statement in a subroutine. A subroutine is a set of 1 to many statements that perform a specific function. Rather than writing the statements many times in a program, the subroutine is used once for the function, saving RAM space. The subroutine is called by a GOSUB. The RETURN statement of a subroutine returns control to the statement immediately following the GOSUB. No line number is required for the RETURN as the BASIC interpreter automatically records the line number after the GOSUB.

RELATED COMMANDS

GOSUB, ON...GOSUB

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*RIGHT$*(string,n)*

EXAMPLES

1000 A$=RIGHT$(B$,4) *get the last 4 characters of B$*
1010 C$=RIGHT$(B$,5) *get the last 5 characters of B$*

DESCRIPTION

RIGHT$ finds the last n characters of a given string. The n parameter may be 0 to 255. The string parameter is a previously defined string. If B$="HEROINE", for example, A$=RIGHT$(B$,4) will set A$="OINE". If n is greater than the length of the specified string, RIGHT$ will return the entire string. A$=RIGHT$(B$,20), for example, returns A$="HEROINE". The n argument may be a constant, variable, or expression. RIGHT$ may be used to process "substrings" where a large string is made up of a number of substrings concatenated together for ease of handling.

RELATED COMMANDS

LEFT$, MID$

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*RND(0)...
*line#...*RND*(integer)...*

EXAMPLES

1000 A=RND(10) *generates a random number from 1 to 10*
1010 IF A=1 THEN PRINT ''STARSHIP MALFUNCTION'' *simulates a chance condition 1 out of 10 times*

DESCRIPTION

RND is a function that generates a pseudo-random number. If the RND(0) form is used, the number is between 0 and less than 1. Typical numbers might be .6789..., .2344..., and 1.2222.... If the RND(N) form is used, where N is not 0, then RND generates a number from 1 to N. If N were 1000, for example, the number generated would range from 1 to 1000 and might typically be 23, 999, 456, 2, 45, etc. Pseudo-random numbers are "repeatable", that is, they produce the same sequence of numbers from a given starting number. A starting number of 23 might always produce the sequence 23, 456, 888, for example. Over a long period, the numbers in the range tend to be evenly distributed; there will be an equal number of 1s, 2s, 3s, 4s, etc.

RELATED COMMANDS

RANDOM

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

*line# ...*ROW*(dummy)...*

EXAMPLES

1000 R=ROW(0)

DESCRIPTION

ROW finds the current row on which the cursor is located and returns the row number. Rows on the Model II are numbered from 0 through 23. The "dummy" parameter is any value enclosed in parentheses; it has no effect on the function. ROW is used along with POS to define the cursor position for word processing and other applications.

RELATED COMMANDS

POS

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

line# RSET *field name=string*

EXAMPLES

1000 RSET NM$=A$ *store addressee name*

DESCRIPTION

RSET is used to place character data into a random-file buffer. The normal sequence of operations establishing a random-file buffer is as follows: Define the fields of the buffer by a FIELD statement. The FIELD establishes the field names in the buffer. The RSET and LSET are then used to store character data in the fields of the buffer. The FIELD statement establishes the size for each buffer field. If the data to be stored by RSET is not as great as this field size, "filler spaces" would be filled on the left. If the field NM$ was 20 characters, the name "SPIRO SMITH" would be stored as "SPIRO SMITH". If data to be stored by RSET is greater than the field size, characters are truncated on the right. The data "SPIRO AGOUPOPOPODOUPOLIS" would be stored as "SPIRO AGOUPOPOPODOUP".

RELATED COMMANDS

FIELD, LSET

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

RUN
RUN *line#*
line# RUN

EXAMPLES

RUN *in command mode starts* BASIC *program from beginning*

RUN 1000 *in command mode starts program from line* 1000

1000 RUN *in program restarts program from beginning*

DESCRIPTION

RUN clears all variables and resets other BASIC program parameters. RUN in the command mode starts the current BASIC program from the beginning. The RUN *line#* form in the command mode starts the program from a specified line number. Note that all variables are cleared before the start occurs. The RUN form within a program restarts the program from the beginning (or a specified line #); it may be used to restart the program on completion of a game or other continuous task.

RELATED COMMANDS

GOTO

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

RUN *"filename"*
RUN *"filename",R*
line# RUN *"filename"*
line# RUN *"filename",R*

EXAMPLES

RUN ""ACCOUNTS/BAS:1"",R *load, keep files open*

DESCRIPTION

RUN loads and executes a BASIC program from disk. Variables are not cleared as is the case with LOAD. If RUN is used without the R option, RUN will close all open files, load the specified program, and execute it. If RUN is used with the "R" option, RUN will will not close open files, and will load and execute the BASIC program. RUN in either form may be used in a BASIC statement during BASIC program execution. The "filename" is a filespec for a BASIC program stored on disk; it conforms to the general requirements for filespecs - name, extension, password, and drive number. RUN may be used in BASIC programs to "chain" programs, allowing one program to call another in a chain of "overlays". One program may utilize file variables from another program when RUN is used instead of LOAD.

RELATED COMMANDS

LOAD

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

SAVE *"filename"*
SAVE *"filename",A*

EXAMPLES

SAVE ""ACCOUNTS/BAS:1"" *save accounts payable*

DESCRIPTION

SAVE saves a BASIC program from RAM to disk. If the "A" option is not used, the program is SAVEd in "compressed format". Compressed format uses special codes for BASIC commands and binary data for line numbers; it is best for economical disk storage. If the "A" option is used, the program is SAVEd in ASCII format. ASCII format is required for subsequent use by the MERGE and APPEND (TRSDOS) commands. ASCII format also allows transfers of disk files for special applications, such as transferring files by data communications. ASCII files take up more disk storage than compressed format. The "filename" is a filespec for a BASIC program stored on disk; it conforms to the general requirements for filespecs - name, extension, password, and drive number.

RELATED COMMANDS

MERGE

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk •

FORMAT

SAVEM *"filename";startaddr,endaddr,execaddr*

EXAMPLES

```
SAVEM ''SORTPR''.&H3000.&H3FFF.
&H3000
```

DESCRIPTION

SAVEM is a Color Computer Disk BASIC command generally used to save a machine-language program in RAM as a disk file. The "filename" parameter is a standard Disk BASIC file name in the name/extension:drive# format. The extension and drive # are optional. If no extension is given, BASIC will use the extension "BIN". If no drive# is given, the standard DRIVE default will be used. SAVEM can be used to save any binary data in RAM whether it is a machine-language program, data, or both. The startaddr parameter specifies the starting address of the data to be saved. The endaddr parameter specifies the end of the data. The execaddr specifies the address of the start of the program, if applicable. The resulting file is stored as a binary file and can be loaded and executed by the LOADM and EXEC commands.

RELATED COMMANDS

EXEC, LOADM

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC •
CC, Disk •

FORMAT

line# SCREEN *type,color set*

EXAMPLES

```
1000 SCREEN 0,1   set text, color set 1
```

DESCRIPTION

SCREEN is used to set the type of display, graphics or text, and to select one of the two color sets available in the Color Computer. The type parameter is either a 0 for a text screen, or a 1 for graphics screen. If a text screen is selected, the text screen starting at location $400 is displayed. This is the "normal" text display mode used to display alphanumeric data. If the graphics mode is selected, the current graphics page is displayed in the current graphics resolution. The current graphics resolution and page are determined by the PMODE command. The "color set" parameter selects one of two color sets. In the text mode, color set 0 is black on green and color set 1 is red on orange. In the graphics mode, the colors depend upon the color set and resolution. (See PMODE.)

RELATED COMMANDS

PMODE

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# SET*(x,y) Model I/III*
line# SET*(x,y,c) Color Computer*

EXAMPLES

1000 SET(RND(127).RND(47)) *set random point I/III*
1010 SET(RND(63).RND(31).3) *set random point to blue (CC)*

DESCRIPTION

Model I/III: SET is used to set one graphics "pixel" to white. There are 6144 pixels, divided up as 128 horizontal elements by 48 vertical elements. Each of the 6144 pixels can be uniquely SET. The x coordinate specifies the horizontal position of 0-127. The y specifies the vertical position of 0-47.
Color Computer: SET is used to set one graphics "pixel" to a specified color, c. There are 2048 pixels, divided up into 64 horizontal elements by 32 vertical elements. The x coordinate specifies the horizontal position of 0-63. The y coordinate specifies the vertical position of 0-31. The c parameter is a color code of 0 through 8 (black, green, yellow, blue, red, buff, cyan, magenta, orange).

RELATED COMMANDS

CLS. POINT. RESET

SYSTEM

J, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*SGN*(expression)...*

EXAMPLES

1000 IF SGN(X)=0 GOTO 2000 ELSE IF SGN(X)=1 GOTO 3000 ELSE GOTO 4000
goto 2000 *if X=0*, 3000 *if X positive, or* 4000 *if X negative*

DESCRIPTION

SGN is a sign function. It finds the sense of a constant, variable, or expression. The argument must be enclosed within parentheses. If the argument is negative, SGN returns a -1; if the argument is 0, SGN returns a 0; if the argument is positive, SGN returns a +1. SGN is a convenient replacement for code such as:

```
1000 IF X<0 THEN A=-1
1010 IF X=0 THEN A=0
1020 IF X>0 THEN A=+1
```

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	●
I, Disk	●
II	●
III, LVL I	
III, LVL III	●
III, Disk	●
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

*line#...*SIN*(expression)...*

EXAMPLES

1000 C=SIN(Z+3.14159/2) *sets variable C equal to sine of X+pi/2 (in radians)*
2000 ND=SIN(X*.01745329) *sets variable ND equal to sine of X (in degrees)*

DESCRIPTION

SIN finds the sine of a given constant, variable, or expression. The quantity is assumed to be in radians (180/pi degrees). SIN is a "function" and may be used anywhere within a BASIC statement as long as the argument is enclosed within parentheses. Multiply by .01745329 to convert degrees to radians. Standard trigonometric rules apply in regard to the sign of the result.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	●
CC, Ext BASIC	●
CC, Disk	●

FORMAT

SKIPF
SKIPF *"filename"*

EXAMPLES

SKIPF ''MYPROG'' *skip over* MYPROG

DESCRIPTION

SKIPF is used to skip over an indicated file on cassette. Executing SKIPF with a filename will cause BASIC to search for the file name and position the tape after the end of file. It is therefore positioned to read the next file after "filename". Executing SKIPF without a filename will cause BASIC to skip the next file on cassette and position the tape after the end of the file, ready to read the next file.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC •
CC, Ext BASIC •
CC, Disk •

FORMAT

line# SOUND *freq,duration*

EXAMPLES

```
1000 FOR I=1 TO 255 set frequency loop
1010 SOUND I,2 output tone
1020 NEXT I loop
```

DESCRIPTION

SOUND outputs a tone to the TV speaker. The frequency of the tone is specified by a "freq" count of 1 to 255. Middle C corresponds roughly to a count of 89. The remaining counts range roughly over four octaves; the lower the count, the lower the note. The frequency count is "linear"; a count of 1/2 the value of another count is 1/2 the frequency. The duration value of 1 through 255 determines the duration of the tone. Each count is roughly 1/16th of a second, making the range of durations 1/16th second to 16 seconds. SOUND can be used to output warning tones or to play musical notes in songs or games.

RELATED COMMANDS

PLAY

SYSTEM

I, LVL I
I, LVL II
I, Disk
II •
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line# ...SPACE$*(expression)*...

EXAMPLES

```
1000 A$=''NAME''+SPACE$(23)
+ ''ADDRESS''
```

DESCRIPTION

SPACE$ returns a string of spaces. It is logically equivalent to STRING$(" ",n), where n is the number of characters to return. The constant, variable, or expression for SPACE$ must be a numeric value from 0 through 255. Spaces (blanks) are commonly used in PRINT or LPRINTing reports and other text processing. SPACE$ provides a convenient way of generating spaces.

RELATED COMMANDS

STRING$

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...SPC*(expression)*...

EXAMPLES

```
1000 PRINT ''NAME'' SPC(23)
''ADDRESS''
```

DESCRIPTION

SPC prints a line of blanks or spaces. SPC does not use string space. The expression parameter must be a numeric value from 0 through 255. The left parentheses must immediately follow the SPC characters. SPC is similar to SPACE$ and can be used with PRINT, LPRINT, and PRINT# to generate spaces or blanks whenever required.

RELATED COMMANDS

SPACE$

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...SQR*(expression)*...

EXAMPLES

1000 C=SQR(A*A + B*B) *find length of triangle side*

DESCRIPTION

SQR is the square root function. It returns the square root of a constant, variable, or expression argument. It can be used anywhere within a BASIC statement as long as the argument is enclosed in parentheses. It is faster than finding the 1/2 power of an argument and should be used in place of this method.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# STOP

EXAMPLES

```
1000 REM STOP HERE TO LOOK AT
VARIABLE I
1010 STOP
```

DESCRIPTION

STOP is used to temporarily stop BASIC program execution. The program may be restarted at the STOP point by the CONT (continue) command. STOP is normally used during program debugging so that intermediate results may be investigated. It is also used as a "breakpoint" to determine if a certain portion of the program is executed. Execution of STOP produces a "BREAK AT XXXXX" message, where XXXXX is the line number. After the stop occurs, variables may be examined by the PRINT or other commands; all intermediate results are left intact.

RELATED COMMANDS

CONT

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*STR$*(expression)...*

EXAMPLES

```
1000 A$=STR$(X) convert X to a string
2000 PRINT STR$(X) print X as a string
```

DESCRIPTION

STR$ converts a numeric constant, variable, or expression to a string. The argument must be within parentheses. In the example above, if X is equal to -34.678, it is converted to the seven-byte ASCII character string of A$="-34.678". If X is equal to 34.678, it is converted to the seven byte ASCII string of A$=" 34.678" with a leading blank for the missing sign. STR$ is used for certain printing or string concatenation functions. The converted value does not have a trailing blank on printing as a numeric value would. Leading zeroes in the numeric value are ignored. A byte is always allocated for the sign and a minus sign or blank is used. An ASCII decimal point is generated in the proper place. The number of fractional characters is somewhat unpredictable and depends upon the value of the expression; trailing zeroes are not generated.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# `STRING$`*(n,"char")*
line# `STRING$`*(n,value)*

EXAMPLES

`1000 A$=STRING$(100,"A")` *create A$="AAAAAA...A"*
`1010 B$=STRING$(50,23)` *create B$=*`CHR$`*(23)+*`CHR$`*(23)...+*`CHR$`*(23)*

DESCRIPTION

`STRING$` is used to create a 1 to 255 character string made up of the same character. The n parameter is the number of characters in the string, from 0 to 255. It may also be a variable or expression that resolves to 0 to 255. The "char" parameter is a single ASCII character that defines the characters in the string. Alternatively, a value of 0 to 255 may be used in place of "char". In the latter case, the equivalent string will be made up of n characters of that value (equivalent to `CHR$`(value)+`CHR$`(value)+...). `STRING$` is used to create strings made up of the same character for screen graphics use, borders, filling dummy data, or other uses.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# `SWAP` *variable1,variable2*

EXAMPLES

`1000 SWAP A,B` *swap variables*

DESCRIPTION

`SWAP` swaps the values of two variables. The variables must have been previously defined (had values assigned to them). Either or both of the variables may be array variables. The variable types of both variables must be the same. `SWAP` can be used in place of code such as "`1000 C=A: A=B: B=C`".

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
SYSTEM
```

EXAMPLES

`SYSTEM` *enter system mode*

DESCRIPTION

`SYSTEM` puts `BASIC` into the System mode. This is a mode in which machine-language files can be loaded from cassette tape. After `SYSTEM` is executed, the `BASIC` interpreter will respond with the prompt `*?`. To load a machine-language program from cassette, position the cassette, and type in the cassette file name, followed by ENTER. `BASIC` will now load the cassette file, flashing asterisks as it does so. After the load, another *? prompt will be displayed. Another machine-language program can now be loaded or control transferred to the machine-language program. In the latter case, type a slash (/) followed by the decimal address for execution, followed by ENTER. If no address is entered after the slash, control will be transferred to the starting address of the file from cassette. (You do not have to know the starting address for a typical cassette load.)

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

```
SYSTEM
SYSTEM "command"
```

EXAMPLES

`SYSTEM` *return to TRSDOS*

DESCRIPTION

`SYSTEM` causes an exit from `BASIC` and a return to TRSDOS. If there is no command, the operation is complete. If there is a TRSDOS command, the command is executed and a return made back to `BASIC`. The command must be enclosed in quotes, unless it is a string expression. If the command involves loading and executing a TRSDOS utility program that involves high memory and "overlay" of BASIC, return will not be made to `BASIC`. `SYSTEM` allows a `BASIC` program to execute a TRSDOS command within the program and then return back to the program. `1000 SYSTEM ''DIR''`, for example, would exit `BASIC`, boot TRSDOS, perform a directory listing, and then return to the next statement after the `SYSTEM` command.

RELATED COMMANDS

None

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*TAB*(expression)...*

EXAMPLES

```
1000 PRINT TAB(25)"BALANCE OFF!"
```

DESCRIPTION

TAB is a special function used with PRINT or LPRINT to "tab over" to a given tab position. The "expression" in TAB must be between 0 and 255. It may be a constant, variable, or expression. The value defines the tab position. When used with PRINT, the cursor is moved to the right to this tab position, and any remaining print items are printed from that point. Valid tab positions for the Model I/III are 0 to 63, for the Model II are 0 to 79, and for the Color Computer are 0 to 31. Values above these will be "modulo" 64, 80, or 32, respectively. When used with LPRINT, the line printer outputs the number of spaces required to effect the tab. TAB cannot move the cursor or line printer print position to the left. If the tab point has already been reached or exceeded, the TAB is ignored.

RELATED COMMANDS

LPRINT, PRINT

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*TAN*(expression)...*

EXAMPLES

`1000 A=TAN(Y+3.14159/2)` *sets variable A equal to tangent of Y+pi/2 (in radians)*

`2000 ND=TAN(X*.01745329)` *sets variable ND equal to tangent of X (in degrees)*

DESCRIPTION

TAN finds the tangent of a given constant, variable, or expression. The quantity is assumed to be in radians (180/pi degrees). TAN is a "function" and may be used anywhere within a BASIC statement as long as the argument is enclosed within parentheses. Multiply by .01745329 to convert degrees to radians. Standard trigonometric rules apply in regard to the sign of the result.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...TIME$...

EXAMPLES

```
1000 PRINT "TIME IS ";TIME$
```

DESCRIPTION

TIME$ returns the current time as a text string. When TRSDOS is started up, the operator may enter the current time. TIME$ returns this information in BASIC. The format of the Model II TIME$ string is HH.MM.SS where HH is the hours, MM is the minutes, and SS is the seconds. The format of the Model I/III TIME$ string is DD/MM/YY HH:MM:SS, where the date is also included.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...TIMER...
line# TIMER=*value*

EXAMPLES

```
1000 TIMER=60   set timer to 12:01
1010 PRINT INT(TIMER/60)  print elapsed
time in seconds
```

DESCRIPTION

TIMER is used to control a built-in "real-time clock" in the Color Computer. The real-time clock increments by one every 1/60th of a second. It counts from 0 through 65,535, at which point it "recycles" back to 0 and begins the counting sequence over again. TIMER can be set to any value by the TIMER=value command; the value represents the starting time in 60ths of a second. After TIMER is set, "reading" TIMER will represent the elapsed time in 60ths of a second, modulo 60. The maximum elapsed time for TIMER is 65,535/60, or about 1092 seconds (18.2 minutes), however, TIMER can be used to control variables that represent any elapsed time by maintaining more precision.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

TROFF
line# TROFF

EXAMPLES

TROFF *turn trace off in command mode*

DESCRIPTION

TROFF turns off the Trace function previously turned off by a TRON command. TROFF is the default condition after BASIC has been initialized.

RELATED COMMANDS

TRON

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

TRON
line# TRON

EXAMPLES

1000 TRON *turn line trace on*
3000 TROFF *turn line trace off*

DESCRIPTION

TRON turns on the BASIC line Trace function. The Trace function executes the program as in normal execution but displays each line number as it is executed within brackets. This trace is useful in following the program flow during program debugging. The SHIFT and @ keys can be pressed simultaneously at any time to stop the display for scrutiny. Pressing any key will restart program execution. Normal display data generated by PRINT or other commands will be interspersed with the Trace line numbers.

RELATED COMMANDS

TROFF

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

```
UNLOAD
UNLOAD drive#
```

EXAMPLES

`UNLOAD 1` *close all open files*

DESCRIPTION

UNLOAD is a Color Computer Disk BASIC command that is a "blanket" CLOSE. It closes all open files for the specified disk drive number. If no disk drive number is specified, UNLOAD closes all open files in the default disk drive (the one specified in the last DRIVE command, or drive 0 if no DRIVE command was ever executed). UNLOAD is primarily used when switching diskettes. The UNLOAD properly closes all open files. Failure to properly CLOSE a disk file may result in loss of all or a portion of the file data on the old or new diskette.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line#...USR*(expression)*...

EXAMPLES

`1000 A=USR(B)` *call machine-language routine*

DESCRIPTION

USR is a function that allows a BASIC program to call a machine-language subroutine. The machine-language subroutine must have been previously loaded into memory and its starting location defined by a special sequence. In the Model I/III this sequence is to POKE the least significant byte of the start address into location 16526 and the most significant byte of the address into location 16527. In the Color Computer the starting address is POKEd into locations 275 (msb) and 276 (lsb). Thereafter, a USR call will cause the BASIC interpreter to transfer control to the code at the machine-language subroutine. The machine-language subroutine will normally return back to the statement following USR. The expression parameter is a constant, variable, or expression that can be resolved down to an integer number. The 16-bit value is passed to the machine-language subroutine under certain conditions. The machine-language subroutine may also return a 16-bit integer value.

RELATED COMMANDS

USRn

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	•
II	•
III, LVL I	
III, LVL III	
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*USRn*(expression)...*

EXAMPLES

`1000 A=USR3(B)` *call machine-language routine*

DESCRIPTION

USRn is a function that allows a BASIC program to call up to 10 machine-language subroutines. The machine-language subroutine must have been previously loaded into memory and its starting location defined by a DEFUSRn. The n parameter in the USRn command matches the n parameter in the DEFUSR command. DEFUSR5, for example, calls the machine-language subroutine defined by DEFUSR5. A USRn call will cause the BASIC interpreter to transfer control to the code at the machine-language subroutine. The machine-language subroutine will normally return back to the statement following the USR. The expression parameter is a constant, variable, or expression that can be resolved down to an integer number. The 16-bit value is passed to the machine-language subroutine under certain conditions. The machine-language subroutine may also return a 16-bit integer value.

RELATED COMMANDS

DEFUSR

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*VAL*(string)...*

EXAMPLES

`1000 A=VAL(PAYABLE$)` *convert to numeric*

DESCRIPTION

The VAL function converts a string, assumed to be a string representing a number, to a numeric value. Typical strings that could be used with VAL are “123.56”, “000100”, and “999.9E-34”. Often, strings that primarily contain numeric data may be represented in string form for input and output operations. VAL provides a way to convert these strings to numeric form for efficient processing. VAL follows these rules in conversion: If the string contains no numeric characters or is null, VAL returns a 0. If the string contains all numeric characters, VAL converts the string to an integer if possible, or to a single-precision number, or to a double-precision number. If the string contains a decimal point, VAL converts the string to a single-or double-precision number. (The Color Computer has only one numeric data type.) VAL ignores alphabetic characters that do not have significance or which it cannot interpret. VAL performs the inverse of the STR$ function.

RELATED COMMANDS

ASC, CHR$, STR$, VAL

SYSTEM

I, LVL I	
I, LVL II	•
I, Disk	•
II	•
III, LVL I	
III, LVL III	•
III, Disk	•
CC, BASIC	
CC, Ext BASIC	•
CC, Disk	•

FORMAT

*line#...*VARPTR*(variable name)...*

EXAMPLES

```
1000 B=VARPTR(A$)
```
get location of A$

DESCRIPTION

VARPTR is a function that finds the address of any BASIC variable. It is primarily used for "parameter" passing to machine-language subroutines called by the USR or USRn commands. If the variable in question is a string variable, VARPTR returns the location of a string parameter block. The first byte of the parameter block is the string length, and the second and third (third and fourth in Color Computer) are the location of the string. If the variable is a numeric variable, VARPTR returns either the location (Models I/III) or a pointer to the value (Color Computer). VARPTR will also return the location of arrays.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	•

FORMAT

```
VERIFY ON
VERIFY OFF
```

EXAMPLES

```
VERIFY ON
```
verify disk writes

DESCRIPTION

VERIFY is a Color Computer Disk BASIC command that turns ON or OFF disk record verification. Records are written out to disk from the disk buffer specified in the OPEN command; a buffer represents one sector's worth of data. When VERIFY is ON, the sector just written is read in to a second buffer and compared with the original data. When VERIFY is OFF, this compare is not done. The verification process is a safeguard against disk I/O errors, but does increase the "overhead" for disk writes. Invalid data will normally be detected on a read, but verification provides detection during the write operation.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk ●

FORMAT

line# `WRITE#`*buf#,item list*

EXAMPLES

```
1000 WRITE#3,A;B;C$
```
output to file

DESCRIPTION

`WRITE#` performs a write to a sequential disk file. The file must have been previously `OPEN`ed. The `OPEN` command sepcifies a buffer for the filename, and this buffer number is used in the `WRITE#` command. `WRITE#` outputs a list of items to the buffer (to the file). The items may be any number of numeric or string variables. All items are transformed into character strings and written to the disk buffer. The `WRITE#` output to the file is similar to the display output of `PRINT`. If commas are used to separate the items, spaces for tabs will be written. If semicolons are used, no spaces will be used between items. String variables should use `CHR$(34)` to bracket the variables with double quotes if the string variables contain delimiters such as commas or semicolons; otherwise string variables can be used in the list as required.

RELATED COMMANDS

None

SYSTEM

I, LVL I
I, LVL II
I, Disk
II ●
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk

FORMAT

line#...(expression) `XOR` *(expression)...*

EXAMPLES

```
1000 IF ((A<29) XOR (B>5)) THEN C=1
```

DESCRIPTION

`XOR` is used as a relational operator and for bit manipulation. In the first use, `XOR` compares two constants, variables, or expressions. If either expression is true, but not both are true, then the `XOR` function is true. In the example above, the expression is true if variable A is less than 2 `OR` variable B is greater than 5. The `THEN` action would be taken if either expression, but not both was true (expression 1 `XOR` expression 2). In the bit manipulation case, `XOR` is used to logically `XOR` integer variable bits, considered to be binary numbers. An `XOR` of binary values produces a 1 for each bit position if either operand but not both has a 1 bit in that bit position. An `XOR` of the two binary values 10100000 and 11001111 would produce a result of 01101111. The `XOR` in this application can be used to test bits, set individual bits, and perform other bit-wise operations.

RELATED COMMANDS

`AND`, `NOT`

SYSTEM

I, LVL I	•
I, LVL II	•
I, Disk	•
II	•
III, LVL I	•
III, LVL III	•
III, Disk	•
CC, BASIC	•
CC, Ext BASIC	•
CC, Disk	•

FORMAT

line# ...↑(expression)...

EXAMPLES

```
1000 T=(1+I)↑Y
```
find amount over Y years

DESCRIPTION

Up arrow is used to represent exponentiation, raising a number to a power. The power may be a constant, variable, or expression. Fractional powers are permitted. In some systems the up arrow prints as a left bracket. The Model II up arrow is SHIFT,6.

RELATED COMMANDS

None

SYSTEM

I, LVL I	
I, LVL II	
I, Disk	
II	•
III, LVL I	
III, LVL III	
III, Disk	
CC, BASIC	
CC, Ext BASIC	
CC, Disk	

FORMAT

line# ...expression \ expression...

EXAMPLES

```
1000 C=A \B
```

DESCRIPTION

Reverse slash (CTRL,9) is a numeric operator that performs an "integer division" on two operands and returns a result. The two expressions involved are converted to two integer operands. An integer division operation divides the first operand by the second operand and finds the quotient. Any fractional part of the quotient is ignored and the integer portion is then returned as the result of the operation. If the first operand is 100, and the second is 44, the result of 100 \44 is the integer portion of 100/44, or 2. This integer division is similar to the INT function except that the two operands here must be in the range of -32768 through +32767.

RELATED COMMANDS

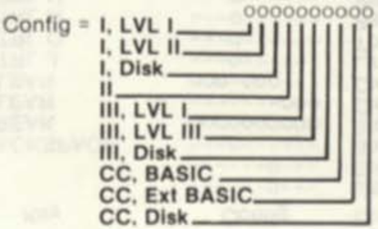
INT

APPENDICES

Special Keys for BASIC

Key	Config	Description
	------o---	List current line
	------o---	Edit current line
BACKSPACE	---o------	Backspace
BREAK	oooooooooo	Stops, sets command mode
CLEAR	-------ooo	Clear screen
CLEAR	ooo-ooo---	Clear and reset 64 char mode
CTRL J	---o------	Line feed without end
CTRL O	---o------	Toggle display function
CTRL R	---o------	Retype current line
CTRL U	---o------	Restart current line
down arrow	------o---	Scroll down during list
down arrow	-oo--oo---	Line feed without end
ENTER	oooooooooo	Terminates input
HOLD	---o------	Halt display during execution
F1, ENTER	---o------	Enter Edit mode for immediate line
left arrow	oooooooooo	Backspace and delete character
REPEAT	---o------	Repeat a single key
right arrow	-oo--ooooo	Space to next tab
SHIFT, @	-oo--ooooo	Halt display during execution
SHIFT, 0	-------ooo	Toggle reverse or lower case
SHIFT, down arrow	------o---	List last program line

SHIFT, left arrow	-oo-oooooo	Delete line, return
SHIFT, right arrow	-oo--oo---	Set 32-char mode
SHIFT, up arrow	-------o---	List first program line
space bar	oooooooooo	Blank
TAB	---o------	Space to next tab.
up arrow	o---o-----	Halt display during execution
up arrow	o---o-o---	Scroll up during list



Config = I, LVL I
I, LVL II
I, Disk
II
III, LVL I
III, LVL III
III, Disk
CC, BASIC
CC, Ext BASIC
CC, Disk

Error Codes

Configuration		Description
1- 1- 1- 1- 1- *- *-*	NF	NEXT without FOR
2- 2- 2- 2- 2- *- *-*	SN	Syntax error
3- 3- 3- 3- 3- *- *-*	RG	RETURN without GOSUB
4- 4- 4- 4- 4- *- *-*	OD	Out of DATA
5- 5- 5- 5- 5- *- *-*	FC	Illegal function call
6- 6- 6- 6- 6- *- *-*	OV	Overflow
7- 7- 7- 7- 7- *- *-*	OM	Out of memory
8- 8- 8- 8- 8- *- *-*	UL	Undefined line
9- 9- 9- 9- 9- *- *-*	BS	Subscript out of range
10-10-10-10-10- *- *-*	DD	Redimensioned array
11-11-11-11-11- *- *-*	/0	Division by 0
12-12-12-12-12- *- *-*	ID	Illegal direct
13-13-13-13-13- *- *-*	TM	Type mismatch
14-14-14-14-14- *- *-*	OS	Out of string space
15-15-15-15-15- *- *-*	LS	String too long
16-16-16-16-16- *- *-*	ST	String too complex
17-17-17-17-17- *- *-*	CN	Can't continue
- -18- - - - -	UF	Undefined user function

Configuration	Description
18-18-19-18-18- - - NR	No RESUME
19-19-20-19-19- - - RW	RESUME without error
20-20-21-20-20- - - UE	Unprintable error
21-21-22-21-21- - - MO	Missing operand
- -23- - - - - BO	Buffer overflow
22-22- -22-22- *- *-* FD	Bad file data
23-23- -23-23- - - L3	Disk BASIC only
-50-50- -51- - -* FO	Field overflow
-51-51- -52- - - IE	Internal error
-52-52- -53- - - BN	Bad file number
-53-53- -54- - - FF	File not found
-54-54- -55- - - BM	Bad file mode
- -55- - - - *-* AO	File already open
-57-56- -58- - - IO	Disk I/O error
- -57- - - - - FE	Undefined Mod II BASIC
- -58- - - - - UE	Undefined error
-61-59- -62- - -* DF	Disk full
-62-60- -63- - - EF	Input past end
-63-61- -64- - - RN	Bad record number
- -62- - - - - NM	Undefined Mod II BASIC

Configuration	Description
- -63- - - - - MM	Mode mismatch
- -64- - - - - UE	Undefined error
-64- - -65- - -	Bad filename
-66-65- -67- *- *-* DS	Direct statement in file
- -66- - - - - FL	Undefined Mod II BASIC
-67- - -68- - -	Too many files
-68- - -69- - -	Disk write protected
-69- - -70- - -	File access denied
- - - - - - -* AE	File already exists
- - - - - - -* BR	Bad record number
- - - - - *- *-* DN	Device number error
- - - - - - -* ER	I/O past end of record
- - - - - - -* FM	Bad file mode
- - - - - - -* FN	Bad file name
- - - - - - -* FS	Bad file structure
- - - - - - *-* IE	Input past end of file
- - - - - *- *-* IO	I/O error
- - - - - - -* NE	Can't find disk file
- - - - - - *-* NO	File not open
- - - - - - -* OB	Out of buffer space

```
- - - - - - -* SE    Set to non-fielded string
- - - - - - -* VF    Verification error
- - - - - - -* WP    Write protected

                                   OOOOOOOO
Configuration is: I, LVL II________|||||||
                  I, Disk___________||||||
                  II_________________|||||
                  III, LVL III________||||
                  III, Disk____________|||
                  CC, BASIC_____________||
                  CC, Ext BASIC__________|
                  CC, Disk_______________|
```

* = mnemonic used in lieu of code

Common ASCII Characters Used in BASIC

CHAR	DEC	HEX
space	32	20
!	33	21
"	34	22
#	35	23
$	36	24
%	37	25
&	38	26
'	39	27
(	40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
:	58	3A
;	59	3B
<	60	3C
=	61	3D
>	62	3E
?	63	3F
@	64	40
A	65	41
B	66	42
C	67	43
D	68	44
E	69	45
F	70	46
G	71	47
H	72	48
I	73	49
J	74	4A
K	75	4B
L	76	4C
M	77	4D

N	78	4E
O	79	4F
P	80	50
Q	81	51
R	82	52
S	83	53
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
	91	5B
	92	5C
	93	5D
	94	5E
	95	5F
	96	60
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A
	123	7B
	124	7C
	125	7D
	126	7E
	127	7F

Decimal/Binary/Octal/Hexadecimal Conversions

DEC	BIN	OCT	HX
0	00000000	000	00
1	00000001	001	01
2	00000010	002	02
3	00000011	003	03
4	00000100	004	04
5	00000101	005	05
6	00000110	006	06
7	00000111	007	07
8	00001000	010	08
9	00001001	011	09
10	00001010	012	0A
11	00001011	013	0B
12	00001100	014	0C
13	00001101	015	0D
14	00001110	016	0E
15	00001111	017	0F
16	00010000	020	10
17	00010001	021	11
18	00010010	022	12
19	00010011	023	13
20	00010100	024	14
21	00010101	025	15
22	00010110	026	16
23	00010111	027	17
24	00011000	030	18
25	00011001	031	19
26	00011010	032	1A
27	00011011	033	1B
28	00011100	034	1C
29	00011101	035	1D
30	00011110	036	1E
31	00011111	037	1F
32	00100000	040	20
33	00100001	041	21
34	00100010	042	22
35	00100011	043	23
36	00100100	044	24
37	00100101	045	25
38	00100110	046	26
39	00100111	047	27
40	00101000	050	28
41	00101001	051	29
42	00101010	052	2A

DEC	BIN	OCT	HX
43	00101011	053	2B
44	00101100	054	2C
45	00101101	055	2D
46	00101110	056	2E
47	00101111	057	2F
48	00110000	060	30
49	00110001	061	31
50	00110010	062	32
51	00110011	063	33
52	00110100	064	34
53	00110101	065	35
54	00110110	066	36
55	00110111	067	37
56	00111000	070	38
57	00111001	071	39
58	00111010	072	3A
59	00111011	073	3B
60	00111100	074	3C
61	00111101	075	3D
62	00111110	076	3E
63	00111111	077	3F
64	01000000	100	40
65	01000001	101	41
66	01000010	102	42
67	01000011	103	43
68	01000100	104	44
69	01000101	105	45
70	01000110	106	46
71	01000111	107	47
72	01001000	110	48
73	01001001	111	49
74	01001010	112	4A
75	01001011	113	4B
76	01001100	114	4C
77	01001101	115	4D
78	01001110	116	4E
79	01001111	117	4F
80	01010000	120	50
81	01010001	121	51
82	01010010	122	52
83	01010011	123	53
84	01010100	124	54
85	01010101	125	55
86	01010110	126	56
87	01010111	127	57
88	01011000	130	58

DEC	BIN	OCT	HX
89	01011001	131	59
90	01011010	132	5A
91	01011011	133	5B
92	01011100	134	5C
93	01011101	135	5D
94	01011110	136	5E
95	01011111	137	5F
96	01100000	140	60
97	01100001	141	61
98	01100010	142	62
99	01100011	143	63
100	01100100	144	64
101	01100101	145	65
102	01100110	146	66
103	01100111	147	67
104	01101000	150	68
105	01101001	151	69
106	01101010	152	6A
107	01101011	153	6B
108	01101100	154	6C
109	01101101	155	6D
110	01101110	156	6E
111	01101111	157	6F
112	01110000	160	70
113	01110001	161	71
114	01110010	162	72
115	01110011	163	73
116	01110100	164	74
117	01110101	165	75
118	01110110	166	76
119	01110111	167	77
120	01111000	170	78
121	01111001	171	79
122	01111010	172	7A
123	01111011	173	7B
124	01111100	174	7C
125	01111101	175	7D
126	01111110	176	7E
127	01111111	177	7F
128	10000000	200	80
129	10000001	201	81
130	10000010	202	82
131	10000011	203	83
132	10000100	204	84
133	10000101	205	85
134	10000110	206	86

DEC	BIN	OCT	HX
135	10000111	207	87
136	10001000	210	88
137	10001001	211	89
138	10001010	212	8A
139	10001011	213	8B
140	10001100	214	8C
141	10001101	215	8D
142	10001110	216	8E
143	10001111	217	8F
144	10010000	220	90
145	10010001	221	91
146	10010010	222	92
147	10010011	223	93
148	10010100	224	94
149	10010101	225	95
150	10010110	226	96
151	10010111	227	97
152	10011000	230	98
153	10011001	231	99
154	10011010	232	9A
155	10011011	233	9B
156	10011100	234	9C
157	10011101	235	9D
158	10011110	236	9E
159	10011111	237	9F
160	10100000	240	A0
161	10100001	241	A1
162	10100010	242	A2
163	10100011	243	A3
164	10100100	244	A4
165	10100101	245	A5
166	10100110	246	A6
167	10100111	247	A7
168	10101000	250	A8
169	10101001	251	A9
170	10101010	252	AA
171	10101011	253	AB
172	10101100	254	AC
173	10101101	255	AD
174	10101110	256	AE
175	10101111	257	AF
176	10110000	260	B0
177	10110001	261	B1
178	10110010	262	B2
179	10110011	263	B3
180	10110100	264	B4

DEC	BIN	OCT	HX
181	10110101	265	B5
182	10110110	266	B6
183	10110111	267	B7
184	10111000	270	B8
185	10111001	271	B9
186	10111010	272	BA
187	10111011	273	BB
188	10111100	274	BC
189	10111101	275	BD
190	10111110	276	BE
191	10111111	277	BF
192	11000000	300	C0
193	11000001	301	C1
194	11000010	302	C2
195	11000011	303	C3
196	11000100	304	C4
197	11000101	305	C5
198	11000110	306	C6
199	11000111	307	C7
200	11001000	310	C8
201	11001001	311	C9
202	11001010	312	CA
203	11001011	313	CB
204	11001100	314	CC
205	11001101	315	CD
206	11001110	316	CE
207	11001111	317	CF
208	11010000	320	D0
209	11010001	321	D1
210	11010010	322	D2
211	11010011	323	D3
212	11010100	324	D4
213	11010101	325	D5
214	11010110	326	D6
215	11010111	327	D7
216	11011000	330	D8
217	11011001	331	D9
218	11011010	332	DA
219	11011011	333	DB
220	11011100	334	DC
221	11011101	335	DD
222	11011110	336	DE
223	11011111	337	DF
224	11100000	340	E0
225	11100001	341	E1
226	11100010	342	E2

DEC	BIN	OCT	HX
227	11100011	343	E3
228	11100100	344	E4
229	11100101	345	E5
230	11100110	346	E6
231	11100111	347	E7
232	11101000	350	E8
233	11101001	351	E9
234	11101010	352	EA
235	11101011	353	EB
236	11101100	354	EC
237	11101101	355	ED
238	11101110	356	EE
239	11101111	357	EF
240	11110000	360	F0
241	11110001	361	F1
242	11110010	362	F2
243	11110011	363	F3
244	11110100	364	F4
245	11110101	365	F5
246	11110110	366	F6
247	11110111	367	F7
248	11111000	370	F8
249	11111001	371	F9
250	11111010	372	FA
251	11111011	373	FB
252	11111100	374	FC
253	11111101	375	FD
254	11111110	376	FE
255	11111111	377	FF