Tektronix®
COMMITTED TO EXCELLENCE

*Please Check for*
*CHANGE INFORMATION*
*at the Rear of this Manual*

# 4052A/4054A

## BASIC and GPIB ENHANCEMENTS

PROGRAMMER'S REFERENCE

**Tektronix, Inc.**
**P.O. Box 500**
**Beaverton, Oregon 97077**

MANUAL PART NO. 070-4383-00
PRODUCT GROUP 14

First Printing JUL 1982
Revised JUN 1983

## MANUAL REVISION STATUS

**PRODUCT: 4052A and 4054A**

This manual supports the following versions of this product: Serial Numbers B020100 and up.

| REV DATE | DESCRIPTION |
|---|---|
| JUL 1982 | Original Issue |
| DEC 1982 | Revised: pages 3-4, 3-7, 3-8, 3-24, and 3-35. |
| JUN 1983 | Revised: page 4-14. |

# CONTENTS

# Section 1

# INTRODUCTION

## PURPOSE

This manual provides information on the BASIC and GPIB features of the 4052A and the 4054A which are not included in the 4052 and 4054. The information in this manual is intended to supplement information in the 4050 Series Graphic System Reference Manual.

## NEW BASIC FEATURES

The most important BASIC enhancements are in the areas of:

- Program presentation — multi-character identifiers, comment tails, OLD/APPEND without remarks.
- Program structuring — subprograms, IF THEN/ELSE/END IF, DO/EXIT IF/LOOP.
- Strings — ALTER, character input, concatenation, string searching, CHR extended.
- Graphics — dashed lines, cross-hatched areas.
- Miscellaneous — MOD operator, logical units (I/O) as expressions.

## NEW GPIB FEATURES

The 4052A/4054A GPIB interface conforms to IEEE Standard 488-1978. It has been modified to provide the following improvements over the 4052/4054:

- Increased GPIB binary data transfer speed.
- Elimination of undesirable state transitions in the GPIB bus management signals.
- Increased GPIB ASCII transfer speed.
- Ability to do data block transfers between the GPIB port and system memory.
- Additional capabilities as standard user accessible features.

# Section 2

# BASIC FEATURES

## VARIABLE NAMES

### Numeric Variables and Subprogram Names

Up to 31 characters can be used in the name of a numeric variable (scalar or array) or a subprogram. The rules for forming legal names are:

1. The first 3 characters cannot be the same as the first 3 characters in a BASIC keyword (PRInt, for example). See the list of BASIC keywords in Appendix A.

2. The first character must be a letter or an underscore.

3. Each following character must be a letter, an underscore, or a digit (0-9).

Lowercase letters are treated the same as uppercase letters.

When a program is LISTed or SAVEd, variable names and subprogram names are displayed with the first character capitalized and the following letters in lowercase.

Defined functions (DEF FN_) — cannot use long multi-character identifiers. Only FNA through FNZ are permitted. The formal parameter to a DEF can have up to 31 letters, just like other numeric variables.

### String Variables

The same rules apply as for numeric variables, with one additional rule:

The last character must be a dollar sign ("$"). It can be the 31st character, but not the 32nd.

### Memory Usage

Each variable name used in the program takes 13 bytes of symbol table space, plus N more bytes if the length of the name (N) is greater than 2. The same amount of symbol table space is taken if a variable is used once or a thousand times in the program. (The original 4050 series BASIC also took 13 bytes for each name.)

## COMMENT TAIL

**Syntax Form:**

[Line number] [any BASIC statement] ! [any characters except CR]

**Descriptive Form:**

[Line number] [any BASIC statement] ! [program documentation comment]

### Purpose

The exclamation character can be used to add a comment to the end of any statement, or as a separate statement. For example:

```
100 X=0      ! initialize X
110          !This can be done instead of REM
```

LIST and SAVE produce lines with the exclamation character 2 spaces after the end of the statement. If it appears as a separate statement, then it is put at the 4th position after the line number. Therefore, instead of "    REM" you get "    !" (with the ! character in the same space as the M in REM).

## FORMATTED LIST

**Syntax Form:**

[Line number] PRI @ 37,19: numeric expression

**Descriptive Form:**

[Line number] PRINT @ 37,19: indent value

### Explanation

The list command generates a listing of a BASIC program and optionally formats the program in the following manner:

Indent after the following statements:

SUB
IF (block IF only)
DO
FOR
EXIT IF
ELSE

Cancel indent before the following statements:

ENDSUB
END IF
LOOP
NEXT
EXIT IF
ELSE

The spacing for indenting is user selectable and can be any value from 0 to 10. Spacing of 0 aligns the first character of each line on the left margin. If a line does not fit on the display, short forms of all key words will be used. If it still does not fit, an error is indicated. If the indent value is from one to ten, then that many blanks will be printed for each level of indenting based on the program being listed. Statements which do not fit on a line will be multi-lined (take multiple lines on the display), each line being indented the same amount.

To select the indent amount, the device address for processor status with secondary address of LIST is used:

```
PRINT@ 37,19:X
```

X is an expression and is evaluated into the range 0 to 10. Values less than 0 get 0, values greater than 10 get 10. This value is used to set the indent count for subsequent LIST commands. When the system is turned on the default value of indent is set to 3.

## LOGICAL UNIT NUMBERS

LUN's are numbers from 0 to 9. The values from 1 to 9 allow you to refer to an OPENed disk file. LUN 0 is special, and refers to an open (via FIND) tape file on the internal tape drive. It is only valid in the commands ON and OFF and the function TYP.

LUNs appear in a number of CALL "..." system commands and in the BASIC statements:

```
PRINT # LUN . . .
INPUT # LUN . . .
READ # LUN . . .
WRITE # LUN . . .
ON EOF (LUN) . . .
OFF EOF (LUN)
TYP (LUN)
CLOSE LUN
OPEN A$; LUN, B$, C$
```

In the original 4050 Series BASIC these statements only allowed LUN to be a numeric constant such as 7 and not a variable such as N. In the 4052A and 4054A they can be any scalar numeric expression.

# Section 3

# BASIC COMMANDS

The new BASIC commands are listed alphabetically using the same syntax form as the 4050 Series Reference Manual.

**Syntax Form**

| | |
|---|---|
| ( ) " " ; : , | Enter each exactly as shown. |
| { } | Enter one of items shown; do not enter braces. |
| [ ] | Optional entry; do not enter brackets. Default values are shown if they exist. |
| . . . | Variable number of items may be entered in the same form as the preceding item (single parameter or group of parameters). |

## ALTER

**Syntax Form:**

$$\text{[Line number] ALT} \begin{Bmatrix}\text{String constant}\\ \text{String variable}\end{Bmatrix}\text{, String variable}$$

**Descriptive Form:**

[Line number] ALTER item to be printed, target variable for result

### Purpose/Explanation

Prints a string of characters on the screen, allows you to edit them with the line-editor keys, and then puts the edited characters back into the target variable when you press RETURN. For example:

```
ALTER "edit this",A$
ALTER Name$,New_name$
ALTER A$,A$
```

Control characters in the printed string appear as underlined letters except CR (↑M) which causes a new line operation. Non ASCII (those above 127) characters have an implementation defined appearance.

## ANGLE Function

**Syntax Form:**

ANG (numeric expr, numeric expr)

**Descriptive Form:**

ANGLE (X-coordinate, Y-coordinate)

### Purpose

The ANGLE function produces the angle between the positive X-axis and the vector from (0,0) to (X,Y).

### Explanation

ANGLE produces what is sometimes called a "2-parameter arc tangent". The result of ANGLE depends upon the current trigonometric mode. In RADIANS mode,

```
-PI < ANGLE(X,Y) <= PI
ANGLE(0,0) is defined to be 0
PI radians is 180 degrees and 200 grads
```

## AREA Function

**Syntax Form:**

ARE (numeric identifier, numeric identifier)

**Descriptive Form:**

AREA (X-array, Y-array)

### Purpose/Explanation

The AREA function calculates the area of a polygonal area defined by an X-array and a Y-array (see HATCH). The value produced is independent of the current WINDOW and VIEWPORT settings. The area is positive if the polygon is defined clockwise, and negative if it is defined counter-clockwise.

AREA works correctly for polygons with holes if two conventions are followed:

1. Connect holes to the surrounding contour with two lines that connect a point on the hole with a point on the surrounding contour.
2. Traverse outside contours in a clockwise direction; traverse holes in a counter-clockwise direction.

Self-intersecting polygons not following this convention will produce unexpected results.

Figure 3-1 shows the right and wrong ways to specify a polygon and use the AREA function. Notice that in Figure 3-1A the polygon is incorrectly specified and CENTROID as well as AREA produces unexpected results.

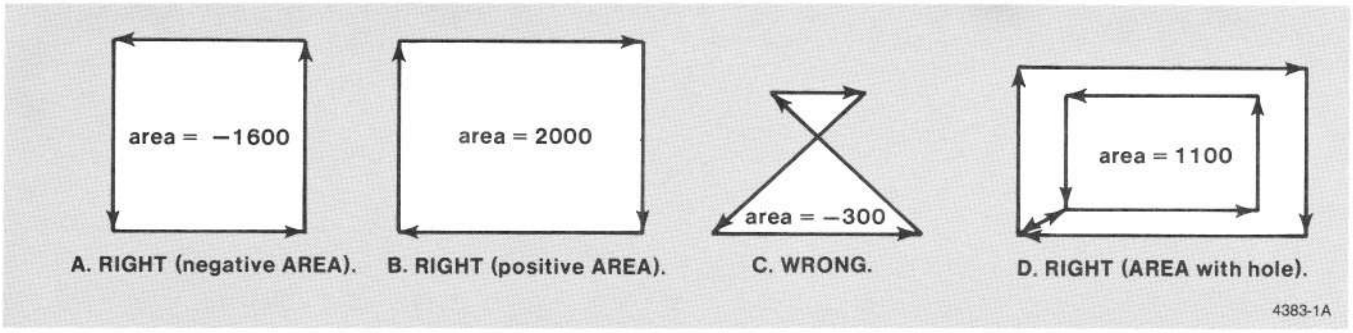
area = −1600
area = 2000
area = −300
area = 1100
A. RIGHT (negative AREA).
B. RIGHT (positive AREA).
C. WRONG.
D. RIGHT (AREA with hole).
4383-1A

Figure 3-1. AREA Function Examples.

The following program was used to generate the examples in Figure 3-1:

```
100 SET DEGREES
110 PAGE
120 DIM X(4),Y(4)
130 READ X,Y
140 DATA 10,50,50,10
150 DATA 10,10,50,50
160 CALL Show(X,Y)
170 READ X,Y
180 DATA 70,70,120,120
190 DATA 10,50,50,10
200 CALL Show(X,Y)
210 READ X,Y
220 DATA 10,50,20,40
230 DATA 60,60,90,90
240 CALL Show(X,Y)
250 DIM X(10),Y(10)
260 READ X,Y
270 DATA 60,60,120,120,60,70,110,110,70,70
280 DATA 60,95,95,60,60,65,65,90,90,65
290 CALL Show(X,Y)
300 END
310 SUB Show(X,Y)
320       FOR I= 1 TO UBOUND(X,1)
330             Ip1= (I MOD UBOUND(X,1))+ 1
340             CALL Arrow(X(I),Y(I),X(Ip1),Y(Ip1))
350       NEXT I
360       CENTROID X,Y,Xc,Yc
370       MOVE Xc,Yc
380       PRINT "    area= ";AREA(X,Y)
390 END SUB
400 SUB Arrow(X1,Y1,X2,Y2)
410       MOVE X1,Y1
420       DRAW X2,Y2
430       ROTATE ANGLE(X2-X1,Y2-Y1)+ 170
440       RDRAW 3,0
450       ROTATE ANGLE(X2-X1,Y2-Y1)+ 190
460       MOVE X2,Y2
470       RDRAW 3,0
480 END SUB
```

## ASC Function

**Syntax Form:**

ASC (simple string [,numeric expr])

**Descriptive Form:**

ASC (source string [,location of character])

### Purpose/Explanation

The ASC function returns a decimal number corresponding to the specified character in the string. If the second argument (location) is not specified, then it defaults to 1. The location is the index in the string of the character whose decimal value (0 to 255) will be produced. A location <= 0 or > LEN(source string) will be an error.

## BINARY OPERATIONS

**Syntax Form:**

```
[line number] CAL "BITAND", simple string, simple string, string variable
[line number] CAL "BITOR", simple string, simple string, string variable
[line number] CAL "BITXOR", simple string, simple string, string variable
[line number] CAL "BITCMP", simple string, string variable
[line number] CAL "BITROT", simple string, numeric expression, string variable
[line number] CAL "BITSHI", simple string, numeric expression, string variable
[line number] CAL "BITTES", simple string, numeric expression, numeric variable
[line number] CAL "BITSET", string variable, numeric expr, numeric expr
```

**Descriptive Form:**

```
[Line number] CALL ""BITAND", In1$, In2$, Out$
[Line number] CALL ""BITOR", In1$, In2$, Out$      !Inclusive OR
[Line number] CALL ""BITXOR", In1$, In2$, Out$     !Exclusive OR
[Line number] CALL ""BITCMP", In0$, Out$           !Bit complement
[Line number] CALL ""BITROTATE", In0$, distance, Out$
[Line number] CALL ""BITSHIFT", In0$, distance, Out$
[Line number] CALL ""BITTEST", In0$, bit number, bit value returned
[Line number] CALL ""BITSET", Inout$, bit number, bit value to set
```

### Purpose

Each of these statements allow a string of N characters to be treated as a sequence of 8*N bits. "BITAND", "BITOR", and "BITXOR" will treat as zero filled (on the left) In1$ or In2$, if they are not the same length. Null strings are allowed as input to all the statements except "BITSET" and "BITTEST".

### Explanation

"BITAND" performs a bit by bit logical AND of In1$ and In2$, and puts the result into Out$ (which can be either In1$ or In2$).

"BITOR" performs a logical inclusive OR (same concept as the OR operator of BASIC).

"BITXOR" performs a logical exclusive or (XOR):

0 XOR 1 is 1
1 XOR 0 is 1
0 XOR 0 is 0
1 XOR 1 is 0

"BITCMP" complements every bit in the string:

0 becomes 1
1 becomes 0

Out$ can be the same string as In0$.

"BITROTATE" rotates the entire sequence of bits by the specified number of bits in the distance argument. A distance less than zero causes a right rotate; a distance greater than zero causes a left rotate. For example:

Rotation distance = 2 — 11110000 becomes 11000011

Rotation distance = -2 — 11110000 becomes 00111100

When distance > 8*LEN(In0$), the result is the same as if the string were rotated by the value (distance) MOD > (8*LEN"ABC") and 32 MOD 24 gives 8. The expression is equivalent to CALL "BITROT", "ABC", 8, L$.

"BITSHIFT" shifts (with zero fill) an entire sequence of bits by the specified number of bits. This works the same as "BITROTATE", except you get zero fill. For example:

Shift distance = 2 — 11111111 becomes 11111100

Out$ can be the same string as In0$.

"BITTEST" allows you to examine the value of any bit in the string. A numeric value of 0 or 1 is returned.

"BITSET" allows you to set any bit in the string to be 0 or 1. The numeric value passed in is treated as logically true or false, just as in the IF statement and NOT function. For example ABS(. . .) >_ 0.5 produces 1, anything else produces 0.

*NOTE*

*For BITROTATE and BITSHIFT, the absolute value of the specified number of bits must be less than 2E16 — 1. The same is true for the bit number in BITSET and BITTEST. For BITROTATE, the target string(string variable) may not be the same as the source string (simple string). For BITSET and BITTEST, the bits in the string are numbered from right to left starting with bit number 1. For example, if the string is 3 characters long the bits are numbered 24, 23, 22, . . . 2, 1.*

*Encoding and Decoding Binary strings may be easily accomplished by using the CHR and ASC functions. See the explanation of CHR and ASC elsewhere in this manual.*

## CALL, SUB, LOCAL, END SUB, and "SYMREUSE"

**Syntax Forms:**

[Line number] CAL name [(expression list)]
Line number SUB name [(variable list)]
[Line number] LOC variable list
[Line number] END SUB
[Line number] CAL "SYMREU" [,numeric variable]

**Descriptive Form:**

[Line number] CALL subname [(expr. and REF variables)]
Line number SUB subname [(formal parameters)]
[Line number] LOCAL names of local variables
[Line number] END SUB
[Line number] CALL "SYMREUSE" [,return variable giving bytes of symbol space freed]

*NOTE*

*Items in the CALL expression list and the SUB variable list may be separated by commas or semicolons.*

### Purpose

These statements provide access to named subprograms, with local variables and parameter passing.

CALL transfers control to its matching SUB.

The LOCAL statement(s) after the SUB allocate local variables for the subprogram.

The END SUB transfers program control back to the statement which follows the most recent CALL.

Call "SYMREUSE" can be used after DELETE line,line so that symbol table space can be reused when an APPEND is done.

No other information is considered to be local to a subprogram. Logical unit numbers for disk files, DATA/READ, subroutines, FUZZ, SET TRACE, WINDOW, etc., are all global.

## Explanation

The CALL statement evaluates the expressions in the expression list (a comma or a semicolon between the items is permitted, and they act the same). It then finds the matching SUB name and starts matching the expressions with the formal parameters (one for one match with the variables named in the SUB variable list).

There must be the same number of expressions as formal parameters and their types must match — string to string and numeric to numeric (array or scalar). For example:

```
  100 DIM A(4)
  110 CALL Check("tom",A)                        ! OK
  120 CALL Check(B$,4*X+ 3)                      ! OK
  130 CALL Check("hy")                           ! error
         .     .
         .     .
 1000 SUB Check(A$,BC)
```

If the expression is simply the name of a variable, such as A and B$, then the data is "passed by reference". This means that any access to or assignment to the formal parameter actually uses the referenced variable (A or B$).

If the expression is not simply the name of a variable, such as "tom" and 4*X+ 3, then the value of the expression is "passed by value". This means that the expression is evaluated and then the value is simply "assigned" to the formal parameter.

A formal parameter name may be the same name as a global variable (a variable not local to a subprogram). Therefore, before a formal parameter is actually matched with the argument the current value of the formal parameter is saved on the stack (CALL, GOSUB/RETURN, and FOR/NEXT all use the same stack).

The LOCAL statement(s) follow the SUB statement. When a LOCAL statement is executed the current values of those variables are pushed onto the stack and the variables are set to be undefined. This allows local variables to have the same name as global variables (or local variables in other subprograms).

The LOCAL statement can also be used inside a GO SUB subroutine, a FOR. . .NEXT loop, or a DO. . .LOOP construct. When any of the enclosing constructs terminate, the previous values of the LOCALized variables are restored.

Recursion is an implicit benefit of formal/local/global variable stacking/management. Each invocation of a subprogram gets its own copy of formal and local variables.

**CAUTION**

*Array elements such as A(2) are considered to be expressions, and therefore are passed by value, not reference.*

If you have a global variable V and subprogram S1 is called with a formal parameter or local variable also called V, then the global value of V is inaccessible to anyone until S1 is done. For example, if S1 calls S2 and S2 refers to variable V, then it is getting the value of V in S1, not the global V. Therefore, global data which will be implicitly used by a number of subprograms should be given a unique name (for example, names ending in underscore could be used for this purpose and never be used for formals or locals).

It is possible to mask the value of a subprogram parameter with a LOCAL statement. For example:

```
1000 X=7
1010 CALL Q(X)
          .
          .
          .
2000 SUB Q(Z)
2010 LOCAL X
2020 Z=Z+1
2030 END SUB
```

This program causes an error, "UNDEFINED VARIABLE IN LINE 2020", because the LOCAL statement in line 2010 set X undefined while Z was referencing X. Two possible solutions for this are:

Use a call-by-value — 1010 CALL Q(X+0.0)

Use a local variable whose name does not conflict with the reference parameter. This naming convention can be extended to local variables.

The END SUB statement DELETEs all local variables and pass by value formal parameters in this subprogram. It then pops the stack to restore their former values. The stack is also popped to restore the value of reference formal parameters. When the stack is popped like this, any FORs or GOSUBs which have been done since the CALL are also popped. Finally, END SUB transfers program control back to the statement which follows the CALL.

If execution reaches a line containing a SUB in some fashion other than via CALL, an error results, and program execution stops. If END SUB is executed and there is no "call" on the stack, then an error occurs.

The CALL "SYMREUSE" statement looks through the stack and every line of BASIC code in the memory to see which symbol table entries are still being used. Being used means they are pointed to by code or they are a defined variable or DIMed array or string. Entries which are not being used are freed for reuse (as other symbol table entries) when the next APPEND is done. The number of bytes freed is returned in the numeric variable, if it is present. If APPENDs are done of many different SUBs, each with its own set of local variables (with different names), then "SYMREUSE" will minimize the amount of memory required for the SUBs.

*NOTE*

*CALL and SUB allow a comma or a semicolon between items in the list. You might use a single semicolon to separate input arguments from the output arguments. For example:*

```
CALL Rect_to_Polar (X,Y;R,Theta)

SUB Rect_to_Polar (X,Y;R,Theta)
```

*X,Y are passed into this subprogram and R,Theta are returned. Following this convention makes it easier to read CALL and SUB statements and figure out what they do.*

## CCINPUT

**Syntax Form:**

[Line number] CCI string variable

**Descriptive Form:**

[Line number] CCINPUT target variable

### Purpose/Explanation

The CCINPUT (Conditional Character input) statement examines the type-ahead buffer which holds up to 28 characters from the keyboard. If the type-ahead buffer is empty, then CCINPUT returns a null string. Otherwise, the character code at the front of the buffer is returned (without echoing the character on the screen) and the character returned is eliminated from the buffer. A character code is returned for all keys on the keyboard except PAGE, HOME, BREAK, COPY, and the UDKs (User Definable Keys). All of these keys execute immediately rather than going into the buffer.

The Line Editor keys and some other special function keys return codes with values above 127. The codes above 127 are:

| **LINE EDITOR keys** | **Other keys** |
|---|---|
| 240 — EXPAND | 245 — AUTONUMBER |
| 241 — BACK SPACE | 246 — STEP PROGRAM |
| 242 — SPACE | 247 — AUTOLOAD |
| 243 — CLEAR | 236 — REWIND |
| 244 — RECALL LINE | |
| 176 — COMPRESS | 137 — TAB |
| 177 — RUB OUT ← | 127 — RUBOUT |
| 178 — RUB OUT → | 141 — RETURN |
| 179 — REPRINT | 138 — LF |
| 180 — RECALL NEXT LINE | 136 — BACKSPACE |

*NOTE*

*The 20 UDKs execute, and do not return a code for CCINPUT.*

*The 19 keys on the numeric key pad return the same code as the corresponding keys in the center of the keyboard.*

*The type-ahead buffer holds up to 28 key codes.*

*All other keys generate their normal ASCII character.*

## CENTROID

**Syntax Form:**

[line number] CEN numeric identifier, numeric identifier, numeric variable, numeric variable

**Descriptive Form:**

[line number] CENTROID X-array, Y-array, X-result, Y-result

### Purpose/Explanation

The CENTROID statement calculates the centroid of a polygonal area and returns that (X,Y) coordinate in X-result, Y-result (which can be elements of an array, for example: CENTROID X1,Y1,X2(4),Y2). The polygonal area is defined by the X-array and Y-array (see HATCH). The value produced is independent of the current WINDOW and VIEWPORT settings.

The CENTROID of an area is its center of mass. For example:

```
100 INIT
110 DIM X1(3),Y1(3)
120 DATA 10, 20, 15
130 DATA 40, 40, 80
140 READ X1,Y1
150 CENTROID X1,Y1,X2,Y2
160    ! X2 is now 15
170    ! Y2 is now 51.xxx
```

*NOTE*

*See AREA for comments about self-intersecting polygons.*

**CAUTION**

*CENTROID will generate a divide-by-zero error if the AREA of the polygon is zero.*

## CHR Function

**Syntax Form:**

CHR(numeric expr)

**Descriptive Form:**

CHR (integer)
CHR$ (integer)

*NOTE*
*CHR is permitted only in a LET statement (see LET).*

### Purpose/Explanation

Same as the original CHR except that the integer range has been expanded to 0 to 255, inclusive. The function name can optionally have a $ appended as its last character.

**CAUTION**

*Avoid PRInting the characters between 128 and 255. PRInting one of these characters to the screen or plotter causes problems because their appearance is implementation defined. PRInting one of them to the tape or disk also causes problems because ADE 255 is used for EOF and INPUT strips the high bit off characters. WRITE/READ should be used to store/retrieve strings which contain these characters, or if the string contains CHR(13) (CR) characters.*

## DASH

**Syntax Form:**

[Line number] DAS numeric expr

**Descriptive Form:**

[Line number] DASH dash mask

### Purpose

On the 4054/4054A screen, this sets a hardware dash pattern for displayed vectors. For other display surfaces such as the 4052/4052A screen and plotters, the BASIC interpreter emulates the dash pattern of the 4054 screen. (See the Option 30 Reference Manual.)

### Explanation

The dash mask is an integer between 0 and 255. This is considered to be an 8-bit binary pattern. If a bit is 0 the vector is drawn. The default dash mask is 0, so all 8-bits are 0. This causes solid vectors to be drawn. Execution of an INIT statement returns the mask to its default value.

## DIM

**Syntax Form:**

See the 4050 Series Reference Manual.

### Explanation

DIM can now be used to dimension an array or string larger than its initial dimension. For example:

```
DIM A$(10)
A$= "ABC"
DIM A$(20)
```

has the same effect as:

```
DIM A$(10)
A$= "ABC"
DIM Dummy$(10)
Dummy$= A$
DELETE A$
DIM A$(20)
A$= Dummy$
DELETE Dummy$
```

**CAUTION**

*Doing the following:*

```
100 DIM A$(1)
110 FOR J= 1 TO MEMORY — 1000
120      DIM A$(J)
130 NEXT J
```

*will stop in line 120 with a MEMORY FULL error when memory is half full. This is because DIM A$(J) does this:*

a. *Gets a new block of J bytes of memory (error if there is no such block).*

b. *Copies A$ data into the new block.*

c. *Deletes the old data block.*

Numeric arrays can also be dimensioned larger:

```
DIM A(10), B(2,6), C(7,4)
  .    .      .      .
  .    .      .      .
DIM A(20), B(3,6), C(7,6)
```

Arrays are stored in row-major order. Increasing the dimensioned size adds new floats to the end of the linear form of the array.

**CAUTION**

*This works as expected for one dimensional array and two dimensional arrays in which the number of columns is unchanged. But changing the number of columns (as in array C above) will cause floats from one row to move to another row.*

## DISTANCE Function

**Syntax Form:**

DIS (numeric expression, numeric expression)

**Descriptive Form:**

DISTANCE (X-coordinate(s), Y-coordinate(s))

### Purpose/Explanation

The DISTANCE function calculates the distance around a polygonal area, defined by an X-array and Y-array (see HATCH). DISTANCE (X,Y) has the same effect as:

```
100     ! assume X,Y have N elements
110 Distance = 0
120 For J= 1 TO N−1
130         Distance = Distance + SQR ((X(J)−X(J+1))↑2 + (Y(J)−Y(J+1))↑2)
140 NEXT J
150 Distance = Distance + SQR ((X(1)−X(N))↑2 + (Y(1)−Y(N))↑2)
```

The value produced is independent of the current WINDOW and VIEWPORT settings. If X and Y are scalar, then the DISTANCE(X,Y) is defined as the distance between the origin and (X,Y).

## DO, LOOP, and EXIT IF

**Syntax Form:**

```
Line number DO
Line number EXI IF numeric expression
Line number LOO
```

**Descriptive Form:**

```
Line number DO
Line number EXIT IF numeric expression
Line number LOOP
```

### Purpose

The DO statement marks the beginning of a loop. The LOOP statement transfers control back to the statement following its DO statement. The EXIT IF statement conditionally transfers control to the statement after the LOOP statement.

### Explanation

When DO is executed its line number is pushed onto the stack, much like when a FOR is executed. When LOOP is executed, it pops the stack to find the most recent entry for a DO. It leaves that entry on the stack and transfers control to the first statement after that DO. For example:

```
100 DO
110       PRINT "infinite loop"
120 LOOP
 or
100 DO
110       PRINT "jump out of a loop"
112       A = 1
115       IF A THEN 130
120 LOOP
```

Note that jumping out of the loop leaves information on the stack, much like jumping out of a FOR/NEXT loop.

The EXIT IF statement provides a clean way to get out of a DO/LOOP. The numeric expression is evaluated. If it is logically false (see IF) then execution continues with the next statement. If it is true then the BASIC interpreter looks at succeeding lines for the matching LOOP statement, and transfers control to the statement after it. An error occurs if EXIT IF fails to find the corresponding LOOP, or if DO wasn't previously executed before EXIT IF or LOOP. For example:

```
100 DO
110       PRINT "Enter a starting value (0 to quit) = ";
120       INPUT N
130 EXIT IF N= 0    !"matches" line 230
140       DO
150             IF 2*INT(0.5*N) THEN
160                   N= 0.5*N
170             ELSE
180                   N= 3*N+ 1
190             END IF
200             PRINT "",N
210       EXIT IF N= 1    ! "matches" line 220
220    LOOP    ! "matches" line 140
230 LOOP    ! "matches" line 100
```

Zero or more EXIT IF statements are allowed within a DO/LOOP. The EXIT IF cannot be used to exit a FOR/NEXT. But, of course, a FOR/NEXT can be surrounded by a DO/LOOP. If EXIT IF is used in this manner, then the effect is to exit a FOR/NEXT cleanly:

```
DO
      FOR J= . . . .
            .
            .
      EXIT IF. . . .
            .
            .
      NEXT J
EXIT IF 1.0 ! 1.0 means "always exit"
LOOP
```

## EXCLUDE

**Syntax Form:**

[Line number] EXC numeric expression

**Descriptive Form:**

[Line number] EXCLUDE exclusion level

### Purpose

The EXCLUDE statement sets an internal flag which affects comments in subsequent OLDs and APPENDs (and BOLD, BAPPEND).

### Explanation

You may want to have many comments in a large program, but not have enough memory to run the program with the comments in. The EXCLUDE statement can be used to delete comments in subsequent OLDs and APPENDs. Nothing resets the EXCLUDE level except the EXCLUDE statement. That is, neither INIT nor OLD have any affect on the EXCLUDE level.

The EXCLUDE levels are:

| | |
|---|---|
| EXCLUDE 0 | Leave all comments alone. This is the default value of the flag when the system is powered on. |
| EXCLUDE 1 | Delete all "comment tails" and change REMs to consist of just the REM keyword itself (each such line will take 11 bytes). |
| EXCLUDE 2 | Delete all "comment tails" and the entire REM statements. |

**CAUTION**

*EXCLUDE 2 should only be used if you've written your program to never do GOTO, GOSUB, etc., to a comment line. Also, be careful about SAVEing the code once the comments have been excluded like this.*

*NOTE*

*If a comment tail is used as a separate statement, that is instead of a REM, then EXCLUDE 1 and 2 treat that line as if it were a REM in terms of the rules for EXCLUDE 1 and 2.*

Overlayed programs frequently have each overlay start with a REM, due to the way their overlay manager works. This REM, or any other that you don't want to be affected by EXCLUDE 1 or 2, can simply be changed to an IMAGE statement. For example:

```
1000 REM a comment
     can be
1000 IMAGE a comment              or           1000 IMAGE REM a comment
```

## HATCH

**Syntax Form:**

```
[Line number] HAT [I/O address] numeric identifier, numeric identifier
[Line number] HAT ROT numeric expr
[Line number] HAT SPA numeric expr
[Line number] HAT ALI numeric expr, numeric expr
```

**Descriptive Form:**

```
[Line number] HATCH [I/O address] X-array, Y-array
[Line number] HATCH ROTATE angle
[Line number] HATCH SPACE distance
[Line number] HATCH ALIGN X intercept, Y intercept
```

### Purpose

The HATCH statement crosshatches the polygonal area defined by X-array and Y-array. The type of hatching is controlled by the statements HATCH ROTATE, HATCH SPACE, and HATCH ALIGN. If a DASH pattern has been specified it is also used in hatching.

### Explanation

The X-array, Y-array arguments define a polygonal area, with the last point in X,Y implicitly connected to the first. Corresponding elements of X-array and Y-array are the X and Y coordinates of the vertices of the polygon. For example, to hatch an X shaped pattern across the screen you could enter the following program:

```
100 INIT
110 PAGE
120 DIM X(4),Y(4)
130 DATA 0,130,0,130
140 DATA 0,0,100,100
150 READ X,Y        !read line 130 for X, 140 for Y
160 HATCH X,Y       !use default hatch pattern
170 DRAW X,Y        !draw perimeter
```

The I/O address allows " " mode addressing to hatch to a GPIB device such as a plotter. HATCH is like AXIS, in that it uses MOVE and DRAW secondary addresses.

HATCH does not draw a line around the perimeter of the polygonal area. If a perimeter is desired, it can easily be done in 2 statements:

```
HATCH X,Y
DRAW X,Y
```

The HATCH statement leaves the cursor at X(N),Y(N) (the last point) to make this convenient.

HATCH does allow a polygon to be self-intersecting. But AREA and CENTROID functions may produce unexpected answers for self intersecting polygons. HATCH does not interrupt vectors at connection lines, overlapping vectors connecting detached areas of the polygon, or holes internal to the polygon.

HATCH ROTATE sets the angle of rotation for crosshatched lines drawn by HATCH. The angle is positive counterclockwise with respect to the X axis, in current trigonometric units when HATCH ROTATE is executed (see SET RADIANS, DEGREES, GRADS). The default angle is 0.

HATCH SPACE sets the perpendicular distance between lines drawn by HATCH. The distance is always in GDUs, regardless of the current WINDOW. The default is 1 GDU.

HATCH ALIGN selects an (X,Y) location which affects the lines drawn by HATCH. The hatch lines are considered to be infinitely long, and the lines are aligned so that one of them exactly passes through the (X,Y) location. The (X,Y) location is always in GDUs, regardless of the current WINDOW. The default is (0,0).

*NOTE*

*The HATCH operator takes temporary storage equivalent to three floating point numbers for each coordinate pair in the polygon. Thus, hatching a 10 point polygon takes 240 bytes of memory during the HATCH operation.*

Execution of an INIT statement returns all HATCH parameters to their default values.

## IF, ELSE, and END IF

**Syntax Form:**

```
[Line number] IF numeric expr THE line number
Line number IF numeric expr THE
Line number ELS
Line number END IF
```

**Descriptive Form:**

```
[Line number] IF numeric expr THEN line number
Line number IF numeric expr THEN
Line number ELSE
Line number END IF
```

### Purpose

The "IF. .THEN line" statement conditionally transfers control to the specified line number.

The "IF. .THEN" statement conditionally transfers control to the next statement, or the statement following the ELSE, or if no ELSE then to the statement following the END IF.

The "ELSE" statement transfers control to the statement following the END IF.

The "END IF" statement is ignored when executed by the BASIC interpreter.

## Explanation

For the IF statement, the numeric expression is evaluated, and considered to be logical true if its absolute value is greater or equal to one half (0.5).

The "IF. . .THEN" transfers control to the next statement if the expression is true. If the expression is false then the BASIC interpreter looks at succeeding lines for the matching ELSE or END IF.

For example:

```
100 IF A THEN
110       PRINT "A is TRUE";
120       IF B THEN
130            PRINT "B is also TRUE"
140       END IF    !"matches" IF in line 120
150 ELSE          !"matches" IF in line 100
160       PRINT "A is FALSE"
170 END IF        !"matches" IF in line 100
180 REM
```

If A is true and B is false, then the lines executed are: 100, 110, 120, 150, 180.

If A is true and B is true, then the lines executed are: 100, 110, 120, 130, 140, 150, 180.

If A is false, then the lines executed are: 100, 160, 170, 180.

When an ELSE statement is executed the BASIC interpreter looks at succeeding lines for the matching END IF. An error occurs if no match is found. An error also can occur when an IF. . .THEN is false and a matching ELSE or END IF cannot be found.

## INSIDE Function

**Syntax Form:**

INS (numeric identifier, numeric identifier, numeric expression, numeric expression)

**Descriptive Form:**

INSIDE (X-array, Y-array, X-coordinate, Y-coordinate)

### Purpose

The INSIDE function determines if an (X,Y) point is outside, on, or within a polygonal area. Self-intersecting polygons are permitted here. In general, INSIDE will return "2" wherever HATCH draws lines. Inside will return "1" on connection lines (see HATCH for a description of connection lines).

### Explanation

The polygonal area is defined by an X-array and a Y-array (see HATCH). The INSIDE function produces the following results:

0 if outside the polygonal area
1 if on the polygon (within FUZZ)
2 if inside the polygonal area

*NOTE*

*Both 1 and 2 are logically-true for IF INSIDE(. . .) THEN. . .*

## LET String Operations

**Syntax Form:**

[Line number] [LET] string variable = string expr

A string expr can be:
string item [& string item [&. . . .]]

A string item can be any of:
string literal
string variable
string function — CHR, STR, SEG (TABLE and TRIM)

[Line number] [LET] string variable = REP (simple string, numeric expr, numeric expr)

where simple string is either:
string literal, or
string variable

**Descriptive Form:**

[Line number] [LET] target variable = string expr

Restrictions:

The target variable can appear in the string expression if:

There is more than one "&" operator, and the target variable appears only as the first item in the list of items being concatenated (for example: A$ = A$ & B$ & CHR(4). . . .).

The expression is one of the special forms: A$ = B$ & A$ or A$= ". . ." & A$.

## Purpose/Explanation

Allows multiple concatenations and multiple uses of string functions to occur in one LET statement. For example:

| Legal Statements | Illegal Statements |
|---|---|
| A$ = A$ & B$ | A$ = B$ & C$ & A$ |
| A$ = A$ & B$ & C$ | A$ = CHR(7) & A$ |
| A$ = B$ & A$ | A$ = B$ & SEG(A$,J,K) |
| A$ = A$ & CHR(13) & CHR(10) | |
| A$ = B$ & STR(X) & C$ & SEG(B$,J,K) | |

## MOD Operator

**Syntax Form:**

numeric expr MOD numeric expr

### Purpose/Explanation

MOD is a numeric operator which produces the remainder of a division operation. It has the same precedence as MIN and MAX.

A MOD B is defined to be A—B*INT(A/B)

A SIZE error is produced if B is zero.

For example:

7 MOD 3 is 1  
8 MOD 3 is 2  
9 MOD 3 is 0  
−7 MOD 3 is 2  
7 MOD −3 is −2  
−7 MOD −3 is −1

## RENUMBER

**Syntax Form:**

[Line number] REN [numeric expression [,numeric expression [,line number [,line number in current program]]]]

**Descriptive Form:**

[Line number] RENUMBER [new starting line number [,increment between new line numbers [,starting line number in current program [,final line number]]]]

### Purpose

The RENUMBER statement causes the BASIC interpreter to renumber the lines in the current program. The specified parameters provide directions for the renumber operation. If parameters are not specified, then the BASIC interpreter renumbers all program statements with line numbers greater than 100. These statements are renumbered with an increment of 10 starting with line number 100.

### Explanation

The parameters of the RENUMBER statement specify which statements are to be renumbered and how they are to be renumbered. The parameters are optional and if not specified the BASIC interpreter renumbers the program according to the default parameters (100,10,100). The final line number provides the programmer with the facility to renumber a range of statements in the program without having to renumber from a particular point to the end.

## RND Function

**Syntax Form:**

RND (numeric expression)

### Purpose/Explanation

RND (X) returns a random number between 0 and 1. RND (0) returns a predefined value for each 4050 series instrument, therefore it can be used in a program to determine which 4050 series instrument is being used.

RND (0) returns the following value:

| | |
|---|---|
| 4051 | 0.1... |
| 4052 | 0.70... |
| 4054 | 0.88... |
| 4054 Opt. 30 | 0.50... |
| 4052A | 0.79... |
| 4054A | 0.89... |
| 4054A Opt. 30 | 0.59... |

## RSUM and CSUM Functions

**Syntax Form:**

RSUM (numeric identifier)
CSUM (numeric identifier)

**Descriptive Form:**

RSUM (array)
CSUM (array)

### Purpose/Explanation

The sum of all the rows (or columns) of a 2-dimensional array is formed by using RSUM (or CSUM). For example:

```
DIM A(3,4), B(3), C(4)
A = (. . .) !Assign     values to A
B = RSUM(A)
C = CSUM(A)
B = CSUM(A)   !error, since size of B doesn't match number of columns of A
B = RSUM(C)   !error, since C is a vector (this would work if B were DIMed (4)
because RSUM and CSUM work on vectors)
```

Also see the SUM function.

## SEARCH Function

**Syntax Form:**

SEA (simple string, simple string, numeric expr) where simple string is either a string literal or a string variable

**Descriptive Form:**

SEARCH (string to be searched, rules for the search, starting location for the search)

### Purpose

The SEARCH function searches for and returns the position of the first character (not a substring of characters) which satisfies the rules for the search. The search begins at the specified starting location and proceeds from left to right.

### Explanation

The second parameter specifies the rules for the search. It must be a string of pairs of characters, with non-decreasing ASCII numeric values. For example:

"09AZaz" is correct,
"AZ09az" will be an error, since "0" has ASCII value of 48 and "Z" is 90,
"09A" is also an error, since its length is odd.

A character satisfies the rules of this string if its ASCII numeric value is >= 1st character of a pair and =< 2nd character of a pair (for any of the pairs). For example:

SEARCH(A$,"09",J) — starts at J in A$ and finds the first numeric digit extent in A$ (any of "0123456789").

B$ = CHR(0) & CHR(31) & CHR(33) & CHR(255).

SEARCH (A$,B$,J) — searches for the first non-blank (blank is 32) in A$, starting at J.

SEARCH(A$,"..09AZaz",J) — searches for a period, digit, upper case letter, or lower case letter.

If no character is found which satisfies the rules, then 0 is returned. SET CASE and SET NOCASE have no effect on SEARCH.

## TABLE Function

**Syntax Form:**

TAB (simple string, simple string)

**Descriptive Form:**

Table (source string, translate table)
Table (source string, translate table)

*NOTE*

*TABLE can only be used in a LET statement (see LET).*

### Purpose

The TABLE string function produces a new string from the source string and translate table.

### Explanation

Each character from the source string is treated as an ASCII decimal number. That number plus 1 is used as an index into the translate table. The character found at that position is put into the result string. For example:

The length of the translate table could be 256, and could translate ASCII to EBCDIC (IBM), or visa versa.

```
A$ = CHR(3) & CHR(10) & CHR(11)
B$ = TABLE(A$, "0123456789ABCDEF")
     would assign the result "3AB" into B$
A$ = TABLE(A$,. .) is also legal
```

TABLE (A$,B$) will produce an error if a character from A$ has a decimal value greater than LEN(B$)−1.

The function name can optionally have a $ appended as its last character.

*NOTE*

*Each of the four string functions, STR, CHR, SEG, and REP can also have a $ appended as its last character.*

## TRIM Function

**Syntax Form:**

TRI (simple string)

**Descriptive Form:**

TRIM (source string)
TRIM$ (source string)

*NOTE*

*TRIM can only be used in a LET statement (see LET).*

### Purpose/Explanation

The TRIM string function takes a source string and produces a new string, which is the same as the source string, except that leading and trailing blanks have been removed. For example:

```
B$ = TRIM(" ABC ")
     assigns "ABC" to B$

A$ = TRIM(A$)
     trims blanks from A$, in-place
```

The function name can optionally have a $ appended as its last character.

## UBOUND Function

**Syntax Form:**

UBO ( numeric or string expression, numeric expression )

**Descriptive Form:**

UBOUND ( { numeric variable | string variable | array element | numeric expression | string literal } , numeric expression )

### Purpose

The UBOUND (Upper Bound) function returns size and type information about a variable or array element.

### Examples

```
100 Dim A(15,4)
110 Rem print number of rows of an array
120 Print Ubound(A,1)
130 Rem print number of columns of an array
140 Print Ubound(A,2)
150 Rem print number of elements in a numeric variable of unknown type
160 Print ABS(Ubound(X,1)*Ubound(X,2))

190 Rem If A$ is not currently DIM'ed then DIM it
200 If Ubound(A$,1)<>0 Then 220
210 Dim A$(100)
220 Rem Continue

300 Rem Find the sum of all defined elements in one dimensional array X
310 Total=0
320 FOR J=1 TO Ubound(X,1)
330 IF Ubound(X(J),1)=0 THEN 350
340 Total=Total+X(J)
350 NEXT J
```

## Explanation

The first parameter is a variable (or array element) whose attributes are being queried.

The second parameter indicates what kind of information is desired. For arrays, the second parameter allows you to find the upper bound of dimensions 1 and 2 of the array. If the parameter is not a numeric expression which rounds to 1 or 2 (or −1 or −2), then an error is generated.

UBOUND(A,NX) returns the following values:

```
If A is a one-dimensional array:
     NX= 1 returns DIM'ed length of A
     NX= 2 returns −1

If A is a two-dimensional array:
     NX= 1 returns length of 1st dimension
     NX= 2 returns length of 2nd dimension

If A is one or two-dimensional:
     NX= −1 returns the current number of elements in the array.
          Doing LET T= UBOUND(A,−1) is the same as:
               LET A= 1    ! sets every element to 1
               LET T= SUM(T)
     NX= −2 returns the largest number of elements created in the array since it
     was last created. For example:

          DIM A(10)
          DIM A(1000)
          DIM A(4,5)
          LET T= UBOUND(A,−2)    ! sets T to 1000
          LET T= UBOUND(A,−1)    ! sets T to 20 (20= 4*5)

If A is not an array:
     NX= 1 or 2 returns 0 if A is undefined
     NX= 1 or 2 returns −1 if A is defined
     NX= −1 or −2 always returns 1
```

UBOUND(A$,NX) returns the following values:

If A$ is undefined and un-DIM'ed:

all NX return 0

If A$ is undefined but DIMensioned:

NX= 1 or 2 returns −0.2
NX= −1 returns the current DIM of A$
NX= −2 returns the largest DIM of A$

For example:

```
DIM A$(100)
DIM A$(10000)
DIM A$(4)
T= UBOUND(A$, 1)   ! sets T to −0.2
T= UBOUND(A$,−1) ! sets T to 4
T= UBOUND(A$,−2) ! sets T to 10000
```

If A$ is defined:

NX= 1 or 2 returns −1
NX= −1 returns the current DIM of A$
NX= −2 returns the largest DIM of A$

UBOUND(A[E1],NX) or UBOUND(A[E1,E2],NX) (where E1 and E2 are any expression) returns the following values:

NX= 1 or 2 returns 0 if the array element is undefined
NX= 1 or 2 returns −1 if the array element is defined
NX= −1 or −2 always returns 1

UBOUND applied to any other kind of numeric expression, or to a string literal, may not be useful; however, it returns the following:

For other kinds of numeric expressions:

NX= 1 or 2 returns −1
NX= −1 or −2 always returns 1

For string literals:

NX= 1 or 2 returns −1
NX= −1 or −2 returns the length of the literal

Table 3-1 summarizes the results of the UBOUND function for various inputs.

**Table 3-1**

**UBOUND Function Results**

| **Identifier** | **Request** | **Result** |
|---|---|---|
| Numeric Scalar | 1 or 2 | 0 if undefined |
| | | −1 if defined |
| 1 dimension array | 1 | Dimensioned length |
| 2 dimension array | 1 | First dimension (rows) |
| | 2 | Second dimension (columns) |
| 1 or 2 dimension array | −1 | Number of elements in array |
| | −2 | Largest number of elements dimensioned |
| String | −1,−2,1,2 | 0 if undefined |
| | 1,2 | −0.2 if defined |
| | −1 | Current DIM |
| | −2 | Largest DIM |

# Section 4

# GPIB COMMANDS

## ADDRESS LIST ARRAYS

The syntax for <address list> is as follows:

```
<address list>             = <address> [ ; <address> ] . . .

<address>                  =     <primary address>
                                 <extended address>
                                 <address array>
                                 <extended address array>

<primary address>          = numeric expression

<extended address>         = numeric expression , numeric expression

<address array>            = one dimensional array

<extended address array>   = two dimensional array
```

Arrays of addresses have the characteristics described in the following paragraphs.

One dimensional arrays address devices using only primary addresses.

Two dimensional arrays address devices which implement extended talker or extended listener functions. These functions require both primary and secondary addresses. In these arrays, element A(i,1) is the primary address for device i, and element A(i,2) is the secondary address for device i. Other columns such as A(i,4) are ignored.

Negative values in an address array are ignored. Undefined values generate error number 36.

In a two dimensional array, if the primary address is negative, then neither primary nor secondary address is sent.

The values in the array should be in the range 1 to 30 for primary addresses, and 0 to 30 for secondary addresses. If a variable is not in this range, then error 66 is generated. The exception is that 4050 series instruments ignore 32 as a secondary address. Undefined addresses generate error number 36. The correct bias to form a Talk, Listen or secondary address is added by each command.

Non integer addresses are first rounded to integers.

In commands where <address list> is optional, if an address list is specified, then the bus configuration is cleared upon completion of the command by sending untalk and unlisten (UNT and UNL) messages. If no address list is specified when it is optional, then it is assumed that the talker or listener have been previously configured, and they are not unconfigured upon completion. The exception to this rule is for the TALK and LISTEN commands, whose function would be defeated by clearing the bus configurations.

## CONFIGURE

**Syntax Form:**

[Line Number] CAL "CONFIG" [ , numeric expression ] , numeric variable ; array variable

**Descriptive Form:**

[Line Number] CALL "CONFIGURE" [ , timeout ] , code ; address(es)

### Purpose

Returns in the array variable an address list which represents the addresses of all active devices on the bus. The address list is in the form described under Address List Arrays. For example:

```
100 Dim A(15,2)
110 Call "Config",E;A
120 If E Then
130 GoSub 2000
140 End If

2000 Print "No devices active on bus"
2010 Return
```

## Explanation

The method used to determine whether a device is present on the bus at a given address is to send the UNL message followed by a listen address. If a device is present at that address, it asserts the NDAC line when ATN is unasserted. Because there is no specification in the IEEE-488 standard, concerning the length of time a device may continue to assert NDAC after ATN goes false when the device is not a listener, a timeout value is required.

The timeout value is specified in milliseconds, and must round to a value within the range 0 to 65535. Values outside of this range generate error number 96. This parameter is optional, and if not present the timeout value will be 1 mS.

If the array variable is one dimensional, then only primary addresses are tested. If the array is two dimensional, then extended addressing will be used. If the parameter is not an array variable, then error message number 18 will occur.

For extended addressing, a two step addressing scheme is used. First, each primary address is tested. For any device which responds, the secondary address in the array is set to −1. If no device responds to the primary address, then sequential secondary addresses are sent out, and if a device responds, the address is put into the array.

If the array is not filled, then it is auto-dimensioned to the size corresponding to the number of devices which responded.

The following values are returned in the code parameter:

0 = No occurred.
1 = Array is not large enough to contain address list.
2 = No devices responded. The array is left unchanged.

## IFC (InterFace Clear)

**Syntax Form:**

[Line number] CAL "IFC"

**Descriptive Form:**

[Line number] CALL "IFC"

### Purpose

To clear the GPIB bus. For example:

```
200 Call "IFC"
```

### Explanation

Asserts the Interface Clear (IFC) bus management line for a minimum of 5 mS.

## OFF SRQ

**Syntax Form:**

[Line number] OFF SRQ

**Descriptive Form:**

[Line number] OFF SRQ

### Purpose

Disables the service request (SRQ) interrupt from being recognized by the 4050 system. For example:

```
200 OFF SRQ
```

### Explanation

This instruction disables trapping of SRQ bus interrupts. An SRQ interrupt which occurs while trapping is disabled will not cause an error or transfer of control. It differs from the original 4050 series implementation in that errors will not be reported when SRQ interrupts occur while disabled. Power up and INIT default to this condition.

## OFF TIMEOUT

**Syntax Form:**

[Line Number] OFF TIM

**Descriptive Form:**

[Line Number] OFF TIMEOUT

### Purpose

This call disables trapping of GPIB timeouts.

### Explanation

At power up and INIT, timeout trapping is disabled.

## ON SRQ

**Syntax Form:**

[line number] ON SRQ THE numeric variable

**Descriptive Form:**

[line number] ON SRQ THEN line number

### Purpose

Enables trapping of SRQ conditions by BASIC programs, and specifies the line number to branch to when an SRQ occurs. For example:

```
100 ON SRQ THEN 2000

2000 Rem GPIB interrupt handling routine
2010 Return
```

### Explanation

This instruction differs from the original 4050 series instruction in that it enables the SRQ trapping as well as specifying a line number.

When this statement is executed the interrupt capability is re-enabled, the state of the SRQ line is tested and, if asserted at that time, a transfer of control occurs to the line number specified. Specifying an invalid line numbers generates error message number 51.

At power up and INIT, SRQ trapping is disabled.

## ON TIMEOUT

**Syntax Form:**

[Line number] ON TIM THE line number

**Descriptive Form:**

[Line number] ON TIMEOUT THEN line number

### Purpose

This command enables you to trap GPIB timeouts and specify the line number of the trap routine. For example:

```
100 Call"TimSet",1,20E-3
110 On Timeout Then 2000

150 Print @ A:S$

200 Off Timeout

2000 Print "GPIB timeout"
2010 Return
```

### Explanation

This instruction enables the GPIB timeout capability, and specifies the line number of the trap routine for the case of I/O statements. For the POLL statement, a timeout does not cause transfer of control to the trap routine, but simply advances the poll address.

Once in the trap routine, a RETURN statement transfers control to the statement following the one being executed when the timeout occurred.

Specifying an invalid line number generates error number 51.

## POLL

**Syntax Form:**

[Line number] POL numeric variable , numeric variable ; < address list>

**Descriptive Form:**

[Line number] POLL index , status ; address(es)

### Purpose

Issues a serial poll to the devices in the address list. A status byte is returned from the device requesting service, or the last device which responded to the poll. For example:

```
100 Dim A(15)
110 On SRQ Then 2000
     .
     .
     .
200 On SRQ Then 3000
     .
     .
     .
300 Rem    Get status from device 6
610 Poll i,s;6
     .
     .
     .
2000 Call "Config",E;A
2010 Poll I,S;A
2030 Rem    Service SRQ
     .
     .
     .
3000 Rem    Poll 3 devices — one uses secondary addressing
3010 Poll I,S;5;3;9,4
3020 Rem    Service SRQ
```

## Explanation

This command is an upward compatible extension of the original 4050 series POLL command. It differs in the following ways:

It adds the capability to use the address lists described in section 4.01 of this document.

A status byte is returned whether or not a device was requesting service.

A user selected timeout can be applied.

The first parameter returns an index into the address list indicating which device the status byte is from. If no device responded with RQS bit set in the status byte then the index is returned set to zero.

The second parameter is the status byte returned from the device. If no device returns with RQS set, then this parameter will be the one from the last device which did not timeout. If all devices timeout then zero is returned. This allows the POLL statement to be used to get device status even if no SRQ has been asserted.

The rest of the arguments are addresses, and follow the conventions outlined usder Address List Arrays.

The timeout threshold is infinite at power up and after INIT. It may be altered via a CALL"TIMSET" command. If a device does not respond to the poll within the threshold, then the poll continues with the next address.

At the end of the serial poll, the Serial Poll Disable (SPD) and Untalk (UNT) messages are sent.

## RENOFF

**Syntax Form:**

[Line Number] CAL "RENOFF"

**Descriptive Form:**

[Line Number] CALL "RENOFF"

### Purpose

Unasserts the Remote ENable (REN) line.

### Explanation

This is the only way to unassert the REN line under program control. (In the event of a source handshake error (GPIB error message 69) the REN line is cycled for 100 ms minimum.) As long as REN is false, devices will stay in the Local State (LOCS). The REN line is asserted on power up and INIT.

## RENON

**Syntax Form:**

[Line Number] CAL "RENON"

**Descriptive Form:**

[Line Number CALL "RENON"

### Purpose

Asserts the Remote ENable (REN) line.

### Explanation

The REN line is also asserted on power up and INIT.

## TIMSET

**Syntax Form:**

[Line number] CAL "TIMSET" , numeric expression [ , numeric expression ]

**Descriptive Form:**

[Line number] CALL "TIMSET" , I/O time threshold [ , POLL time threshold ]

### Purpose

This call allows time thresholds to be set for use with the ON TIMEOUT statement. For example:

```
100 Rem    Set I/O timeout to 50 mS. and leave POLL timeout as is
110 Call"TimSet",50E−3

200 Rem    Set I/O timeout to 1.5 S., Poll timeouts to 90 mS.
210 Call"TimSet",1.5,90E−3

300 Rem    Leave I/O timeout as is, set Poll timeout to 50 mS.
310 Call"TimSet",−1,50E−3
```

### Explanation

The times are specified in seconds, and are rounded to the nearest millisecond. A value of zero indicates an infinite timeout, and is the default on power up and after INIT for I/O. The POLL default timeout is 100 ms.

*NOTE*

*Maximum time that can be specified is 65.535 seconds. Larger values than this will result in an error message.*

The first threshold applies to I/O transfers. The second threshold applies only to the POLL statement. If the second parameter is not present, or if either is negative, then that threshold is not affected.

The actual time threshold is only approximate. The actual time may be much larger than the number specified, especially if many interrupts occur from the keyboard, display, etc.

# Appendix A

## BASIC KEYWORDS

ABS
ACOS
ACS
* ALIGN
ALL
* ALTER
AND
* ANGLE
APPEND
* AREA
ASC
ASIN
ASN
ASSIGN
ATAN
ATN
AXIS

BLINK
BRIGHTNESS

CALL
CASE
* CCINPUT
* CENTROID
CHARSIZE
CHR
* CHR$
CLOSE
COPY
COS
CREATE
* CSUM
CURSOR

DASH
DATA
DEF
DEGREES
DELETE
DET
DIM
DIRECTORY
* DISTANCE
* DO
DRAW

* ELSE
END
EOF
EOI
* EXCLUDE
* EXIT
EXP

FIND
FIX
FNA
FNB
FNC
FND
FNE
FNF
FNG
FNH
FNI
FNJ
FNK
FNL
FNM
FNN
FNO
FNP
FNQ
FNR
FNS
FNT
FNU
FNV
FNW
FNX
FNY
FNZ
FONT
FOR
FULL
FUZZ

GIN
GO
GOSUB
GRADS

* HATCH
HOME

IF
IMAGE
INIT
INPUT
* INSIDE
INT
INV

KEY
KILL

LEN
LET
LGT
LIST
* LOCAL
LOG
* LOOP

MARK
MAX
MEMORY
MIN
* MOD
MOVE
MPY

NEXT
NOCASE
NOKEY
NORMAL
NOT

OF
OFF
OLD
ON
OPEN
OR

* = New keyword in 4052A and 4054A.

| | | | |
|---|---|---|---|
| PAGE | RINIT | SIN | TRACE |
| PI | RMEMORY | SIZE | * TRIM |
| POINTER | RMOVE | SPACE | * TRIM$ |
| POLL | RND | SQR | TRN |
| POS | ROPEN | SRQ | TYP |
| PRINT | ROTATE | STEP | |
| | RREPLACE | STOP | * UBOUND |
| RADIANS | RSPACE | STPOINT | UNIT |
| RAPPEND | * RSUM | STR | USING |
| RBYTE | RUN | * STR$ | |
| RCLOSE | | * SUB | VAL |
| RDELETE | SAVE | SUM | VIEWPORT |
| RDRAW | SCALE | | VISIBILITY |
| READ | * SEARCH | * TABLE | |
| REM | SECRET | * TABLE$ | WAIT |
| RENUMBER | SEG | TAN | WBYTE |
| REP | * SEG$ | THEN | WINDOW |
| * REP$ | SET | * TIMEOUT | WRITE |
| RESTORE | SGN | TLIST | |
| RETURN | SIGN | TO | |

* = New keyword in the 4052A and 4054A.

# Appendix B

# ERROR MESSAGES

| Message Number | Error Message |
|---|---|
| 0 | A firmware failure has occurred. Turn OFF the power switch and wait five seconds before turning it ON again.<br>Example:<br>Loading into the 4051 a program which contains commands available only in the 4052/4054 Graphic Systems. |
| 1[a] | An arithmetic operation has resulted in an out of range number.<br>Example:<br>1/1.0E−308 |
| 2[a] | A divide by zero operation has resulted in an out of range number.<br>Example:<br>4/0<br>Or an attempt was made to do MOD 0 (4052A/4054A only). |
| 3[a] | An exponentiation operation has resulted in an out of range number.<br>Example:<br>5↑1.0E+ 300 |
| 4[a] | An exponentiation operation involving the base e has resulted in an out of range number.<br>Example:<br>EXP (1.0E+ 234) |
| 5[a] | The parameter of a trigonometric function is too large; that is, the variable N in the statement A= SIN(N*2*PI) is greater than 65536.<br>Example:<br>A= SIN(4.2E+ 5) when the trigonometric units are set to RADIANS. |
| 6[a] | An attempt has been made to take the square root of a negative number. The positive square root is returned by default.<br>Example:<br>SQR (−4) |

[a] **This error is caused by a math operation which produces a predefined out of range number. This error condition can be handled by the BASIC program without terminating program execution. Refer to the ON. . .THEN. . . statement in the 4050 Series Graphic System Reference Manual for details.**

| Message Number | Error Message |
|---|---|
| 7 | The line number in the program line is not an integer within the range 1 to 65535.<br>Example:<br>`0 REM THIS IS AN INVALID LINE NUMBER` |
| 8 | The matrix arrays are not conformable in the current math operation; that is, they are not of the same dimension and/or do not have the same number of elements.<br>Example:<br>`INIT`<br>`DIM A(2),B(2),C(3)`<br>`A=1`<br>`B=2`<br>`C=A+B`<br>Or an illegal operation was attempted in CSUM or RSUM which resulted in a shape error (4052A/4054A only). |
| 9 | A previously defined numeric variable can not be dimensioned as an array variable without deleting the numeric variable first.<br>Example:<br>`INIT`<br>`B=3`<br>`DIM B(2,2)` |
| 10 | There is an error in the subscript of a variable due to one of the following:<br>1. A numeric variable can't be subscripted.<br>2. A subscript is out of range.<br>Example 1: `INIT` `DIM A(2,2)` `A(2,3)=5`<br>Example 2: `INIT` `B=3` `PRINT B(4)` |
| 11 | An attempt has been made to use an undefined DEF FN function. |
| 12 | There is a parameter error in the CALL statement to a ROM pack.<br><br>Or the target string is the same as the string to be rotated in BITROTATE, or the absolute value of the bit number to rotate or shift is greater than (2.0E+16)−1 (4052A/4054A only). |

| Message Number | Error Message |
|---|---|
| 13 | A WBYTE parameter is not within the range −255 through + 255.<br>Example:<br>WBYTE 300 |
| 14 | A parameter for the APPEND statement is invalid. |
| 15 | An attempt has been made to APPEND to a nonexistent line number. |
| 16 | There is an invalid parameter in the FUZZ statement.<br>Example: FUZZ −10 |
| 17 | There is an invalid parameter in a RENUMBER operation due to one of the following:<br>1. The first or third parameter is not a line number within the range 1 through 65535.<br>2. The increment (second parameter) is not within the range 1 through 65535 or is so large that out of range line numbers are generated during the RENUMBER operation.<br>3. Statement replacement or statement interlacing will occur if the RENUMBER operation is attempted.<br><br>This error may occur during an APPEND operation. |
| 18 | Not used. |
| 19 | There is an invalid parameter in a GO TO, FOR, or NEXT statement.<br>Example:<br>500 FOR I= 1 to 20 where I has been previously defined as an array variable. |
| 20 | The logical unit number specified in the statement is not within the range 0 through 9.<br><br>100 ON EOF (10) THEN 500 |

| Message Number | Error Message |
|---|---|
| 21 | The assignment statement is invalid because of one of the following:<br>1. An attempt has been made to assign an array to a numeric variable.<br>2. Two arrays in the statement are not conformable (not of the same dimension and/or do not have the same number of elements).<br>3. An attempt has been made to assign a character string to a string variable and the character string is larger than the dimensioned size of the variable. |
| 22 | There is an error in an exponentiation operation because the base is less than 0 and the exponent is not an integer less than 256.<br>Example:<br>−10↑257.5 |
| 23 | An attempt has been made to take the LOG or LGT of a number which is equal to or less than 0.<br>Example:<br>LOG (−1) |
| 24 | The parameter of the ASN function or the ACS function is not within the range −1 to +1.<br>Example:<br>ASN (2) |
| 25 | The parameter of the CHR function is not within the range 0 through 127 (4051/4052/4054) or within the range 0 through 255 (4052A/4054A).<br>Example:<br>A$= CHR(257) |
| 26 | Not used. |
| 27 | The parameter is out of the domain of the function.<br>Example:<br>A$= STR(X)<br>where X has been previously defined as an array variable.<br><br>Or an illegal operation was attempted in CSUM or RSUM which resulted in a size error (4052A/4054A only). |
| 28 | A REP function parameter is invalid. |

| Message Number | Error Message |
|---|---|
| 29 | The parameter in the VAL function is not a character string containing a valid number.<br>Example:<br>`A= VAL("Hi")` |
| 30 | The matrix multiplication operation failed because the arrays are not conformable. |
| 31[a] | The matrix inversion failed because the determinant was 0. This error is treated as a SIZE error. |
| 32 | The routine name specified in the CALL statement can not be found.<br>Example:<br>CALL "FIX IT" where the routine "FIX IT" resides in a ROM pack which is not plugged into the System. |
| 33 | Not used. |
| 34 | The DATA statement is invalid because of one of the following:<br>1. There isn't a DATA statement in the current BASIC program.<br>2. There is not enough data in the DATA statement from the present position of the pointer to the end of the statement.<br>3. An attempt has been made to RESTORE the data statement pointer to a nonexistent DATA statement. |
| 35 | The statements DEF FN, FOR, and ON . . . THEN . . . can not be entered without a line number. |
| 36 | There is an undefined variable in the specified line. A numeric variable has not been assigned a value or an array element has not been assigned a value.<br>Example:<br>`INIT`<br>`DIM A(2,2)`<br>`A(1,2) = 4`<br>`PRINT A` |
| 37 | An extended function ROM (Read Only Memory) is required to perform this operation. |

[a] **This error is caused by a math operation which produces a predefined out of range number. This error condition can be handled by the BASIC program without terminating program execution. Refer to the ON. . .THEN. . . statement in the 4050 Series Graphic System Reference Manual for details.**

| Message Number | Error Message |
|---|---|
| 38 | This output operation cannot be executed because the current BASIC program is marked SECRET. |
| 39 | This operation can not be executed because the Random Access Memory is full. Some program lines or variables must be deleted. |
| 40 | Not used. |
| 41 | A SIZE interrupt condition has occurred and an ON SIZE THEN statement has not been executed in the current BASIC program. |
| 42 | A PAGE FULL interrupt condition has occurred. |
| 43 | A peripheral device on the General Purpose Interface Bus is requesting service and an ON SRQ THEN . . . statement has not been executed in the current BASIC program. |
| 44 | The EOI signal line on the General Purpose Interface Bus has been activated and an ON EOI THEN . . . statement has not been activated in the current BASIC program. |
| 45 | A ROM pack is requesting service and the ON UNIT for external interrupt number 1 has not been activated in the current BASIC program. |
| 46 | A ROM pack is requesting service and the ON UNIT for external interrupt number 2 has not been activated in the current BASIC program. |
| 47 | A ROM pack is requesting service and the ON UNIT for external interrupt number 3 has not been activated in the current BASIC program. |
| 48 | The end of the current file has been reached on an I/O device and an ON EOF THEN . . . statement has not been executed in the current BASIC program. |
| 49 | The statement in the specified line is too long. This error situation occurs if an attempt is made to LIST or SAVE a BASIC program which contains a line with more than 72 characters. Sometimes a RENUMBER operation can make a line longer than 72 characters. |

| Message Number | Error Message |
|---|---|
| 50 | The incoming BASIC program contains a line with more than 72 characters. |
| 51 | The line number specified in this statement cannot be found or is invalid. Example: GO TO 500 where the line 500 doesn't exist or PRINT USING 100: where line 100 isn't an IMAGE statement. |
| 52 | Either the specified magnetic tape file doesn't exist or an attempt has just been made to KILL the LAST (dummy) file. |
| 53 | After 10 attempts, the internal magnetic tape unit has been unable to read a portion of the current magnetic tape. The tape head has been positioned after the bad portion in the file to allow the rest of the file to be read. |
| 54 | The end of the magnetic tape medium has been detected. Marking a file longer than the remaining portion of the tape can cause this error. |
| 55 | An attempt has been made to incorrectly access a magnetic tape file. Example: Executing an OLD statement when the tape head is positioned in the middle of a file. |
| 56 | An attempt has been made to send information to a write-protected tape. Remove the tape cartridge, rotate the write-protect cylinder until the black arrow points away from SAFE, insert the tape cartridge, and try the operation again. |
| 57 | An attempt has been made to read to or write to a nonexistent tape cartridge. Insert a tape cartridge into the tape slot and try the operation again. |
| 58 | An attempt has been made to read data which is stored in an invalid magnetic tape format. The tape format must be compatible with the Graphic System. |
| 59 | A program was not found when the OLD statement was executed. |
| 60 | Not used. |

| Message Number | Error Message |
|---|---|
| 61 | An attempt has been made to execute an invalid operation on an open magnetic tape file.<br>Example:<br>Executing a MARK statement with the tape head positioned in the middle of an open data file. |
| 62 | There is a disc file system parameter error. |
| 63 | There is an error in a binary data header, most likely caused by a machine malfunction. |
| 64 | The character string is too long to output in binary format. The length is limited to 8192 characters. |
| 65 | A parity error has occurred in the 4052 or 4054 RAM memory. Although the error is nonfatal (and the message will not be repeated), further operations are unreliable until power has been turned off and back on. In the 4051 this error is not used. |
| 66 | The primary address in the specified line is not within the range 1 through 255. |
| 67 | An attempt has been made to execute an illegal I/O operation on an internal peripheral device.<br>Example:<br>`DRAW    33:50,50` |
| 68 | The diagnostic loader failed. |
| 69 | An input error or an output error has occurred on the General Purpose Interface Bus. Both the NDAC and NRFD signal lines are inactive high, which is an illegal GPIB state. This usually means that there are no peripheral devices connected to the GPIB. |
| 70 | There is an incomplete literal string specification in the format string.<br>Example:<br>`100 IMAGE 6D,5("MARK` |
| 71 | A format string is not specified for the PRINT USING operation. |

| Message Number | Error Message |
|---|---|
| 72 | A format string is too short or not enough matching data is specified.<br>Example:<br>100 IMAGE 6D<br>110 PRINT USING 100: 23,24,25<br>Line 100 should be: 100 IMAGE 3(6D) |
| 73 | There is an invalid character in the format string specified in the PRINT USING statement. |
| 74 | An n modifier in the format string is out of range or is incorrectly used. When used with the E field operator, n modifiers must be positive integers within the range 1 through 11; they must be within the range 1 through 255 when used with the A,D,L,P,T,X, " , ( , and / field operators. |
| 75 | The format string specified in the PRINT USING statement is too long (that is, there are too many data specifiers for the PRINT statement).<br>Example:<br>100 IMAGE 3(6D)<br>110 PRINT USING 100:A,B<br>Line 100 should be: 100 IMAGE 2(6D) |
| 76 | Parentheses are incorrectly used in the format string which is specified in the PRINT USING statement.<br>Example:<br>100 IMAGE 2(6D<br>110 PRINT USING 100:A,B<br>Line 100 should be: 100 IMAGE 2(6D) |
| 77 | There is an invalid modifier to a field operator in the format string which is specified in the PRINT USING statement.<br>Example:<br>100 IMAGE 2(6D),2S<br>110 PRINT USING 100:A,B<br>Line 100 should be: 100 IMAGE 2(6D),S<br>An n modifier is not allowed. |

| Message Number | Error Message |
|---|---|
| 78 | An S modifier is incorrectly positioned in the format string which is specified in the PRINT USING statement. The S modifier must always be positioned at the end of the format string.<br>Example:<br>`100 IMAGE 4D,S,8A`<br>Line 100 should be: 100 IMAGE 4D,8A,S |
| 79 | A comma is incorrectly used in the format string which is specified in the PRINT USING statement.<br>Example:<br>`100 IMAGE 6,D,S`<br>Line 100 should be: 100 IMAGE 6D,S |
| 80 | A decimal point is incorrectly used in the format string which is specified in the PRINT USING statement.<br>Example:<br>`100 IMAGE .3D`<br>`110 PRINT USING 100:812.345`<br>Line 100 should be: 100 IMAGE FD.3D |
| 81 | A data type mismatch has occurred in the PRINT USING statement.<br>Example:<br>`100 IMAGE 6D,6A`<br>`110 PRINT USING 100: "MARY",26`<br>Line 100 should be : 100 IMAGE 6A,6D |
| 82 | A tabbing error has occurred in the format string which is specified in the PRINT USING statement.<br>Example:<br>`100 IMAGE 10A,2T,FD`<br>`110 PRINT USING 100: "ENTER DATA",D`<br>The absolute tab to position 2 specified by 2T in line 100 cannot occur because the cursor has already advanced beyond position 2. The tab specification must be at least 11T in this case. |
| 83 | A number specified in the PRINT USING statement contains an exponent outside the range ± 127.<br>Example:<br>`100 IMAGE FD.3D`<br>`110 PRINT USING 100:8.5E+ 200` |

| Message Number | Error Message |
|---|---|
| 84 | The IMAGE format string was deleted during the PAGE FULL interrupt routine. |
| 85 | A portion of the IMAGE format string was deleted or altered during the PAGE FULL interrupt routine. |
| 86 | A portion of the data specified in the PRINT statement was deleted during the PAGE FULL interrupt routine. |
| 87 | A data item specified in the PRINT USING statement is too large to fit into the print field specified in the format string.<br>Example:<br>`100 IMAGE 5A`<br>`110 PRINT USING 100: "HORSE FEATHERS"`<br>In this example, the string constant "HORSE FEATHERS" is too large to fit into the 5 character field which is specified in line 100. |
| 88 | Not used. |
| 89 | A ROM pack has issued an error message. |
| 90 | Not used. |
| 91 | Not used. |
| 92 | Not used. |
| 93 | Not used. |
| 94 | Not used. |
| 95 | An internal conversion error has occurred because a parameter in the specified statement is negative. |
| 96 | An internal conversion error has occurred because a parameter in the specified statement is greater than 65535. |

*NOTE*

*Messages numbered 97 through 109 apply to the 4052A and 4054A only.*

| **Message Number** | **Error Message** |
|---|---|
| 97 | The Hatch Space has a non positive argument. |
| 98 | The defined polygon cannot be hatched due to insufficient coordinates, or the polygon is insufficient for AREA, INSIDE, or CENTROID. |
| 99 | The parameters to SEARCH are invalid; the rule string is null or not of even length or its values are not incrementing. |
| 100 | The parameter to EXCLUDE is out of range. |
| 101 | The parameter to ASC is not in the range 1 ≤ H parameter ≤ H length of source string. |
| 102 | An error was made in attempting to translate; a character in the source string tried to index outside the translate table. |
| 103 | There is an invalid parameter in the TABLE function. You cannot assign to the translate table while using it. |
| 104 | An assignment has been made to an invalid structure (trying to store a scalar into an array). |
| 105 | An attempt was made to access an undefined subprogram in a CALL statement. |
| 106 | An attempt was made to execute a SUB statement when it was not called. |
| 107 | There was no CALL on the stack when an END SUB was executed. |
| 108 | No END IF can be found to exit an IF due to an ELSE, or no LOOP can be found to exit a DO due to an EXIT IF, or no DO can be found to iterate to from a LOOP, or no ELSE or END IF can be found to exit a false IF condition. |
| 109 | The formal argument in a CALL statement is not compatible with the actual argument in the SUB statement. (For example, trying to pass a string to a numeric variable. |