E · L · E · C · T · R · O · N · I · C · S

MANUFACTURER OF ELECTRONIC TEST EQUIPMENT

BOARDMASTER 4000 & DDS-40

DIAGNOSTIC SYSTEMS

PROGRAMMING MANUAL

A B I Electronics Ltd, Mason Way, Platts Common Industrial Park,
Barnsley, South Yorkshire, S74 9TG, ENGLAND.

Tel: 0226 350145 Fax: 0226 350483 Tlx: 547938 EXPERT G

International Tel: + 44 226 350145 Fax: + 44 226 350483

BoardMaster 4000/DDS-40 Programming Manual - CONTENTS

SECTION 1 - THE TESTBASIC LANGUAGE

1.1 INTRODUCTION

The programming language (TESTBASIC) used on the BoardMaster 4000, DDS-40 and DDS-40XP models is a simple subset of the widely used BASIC language, with additional commands to allow access to the hardware resources of the test system. A detailed description of BASIC is beyond the scope of this manual, but there are many good texts available for those who are not familiar with the language.

If you have used BASIC before, you will have no difficulty in learning the extra commands in TESTBASIC, particularly if you are by now familiar with the operation of the test system. One major limitation of TESTBASIC is its numerical handling capacity - only whole numbers in the range of +/- 32768 can be handled. This is never a limitation when testing ICs, but should be borne in mind if TESTBASIC is used for other functions.

1.1.1 APPLICATIONS OF TESTBASIC

You will occasionally come across ICs that are not included in the standard library of the test system. They may be semi-custom or full custom ICs, or they may be programmable ICs such as PALs or ROMs, and occasionally you may come across an IC that is connected in an "unrecognisable" manner. In these cases you can write your own programs in TESTBASIC to test these ICs. You can execute the programs directly in TESTBASIC using the RUN command, or you can store them on the disk and execute them by simply typing in the appropriate type number. In this way you can make your own additions and extensions to the built-in library.

The applications of TESTBASIC, however, extend beyond IC testing. Simple functional testing can be performed, using the 40 I/O pins to interface to an complete circuit board or even an entire product through any TTL compatible inputs or outputs. A MEASURE command allows analogue input voltages in the range of -10 to +10 volts to be measured, so that the system can act as a data logger with the appropriate sensors and/or transducers at the inputs.

1.1.2 ACTIVATING PROGRAM MODE

To start PROGRAM mode on the DDS-40P or XP press the MODE key until PROGRAM mode is indicated, then press ACCEPT. The message "ENTER SPACE ON EXTERNAL KEYBOARD" will then be displayed. On pressing the space bar on the external keyboard you will see the TESTBASIC sign on message and version number, and the programming language is ready for use. On the BoardMaster 4000 select CUSTOM DEVICE PROGRAM from the main menu and press EXECUTE. At the bottom of the screen the available memory space for your program is displayed. We will now describe very briefly the available commands, but a full description is given later.

1.2 BASIC TESTBASIC

A full description of all commands and functions is given later, but prior to that we will give a brief overview of the TESTBASIC language.

1.2.1 VARIABLES

TESTBASIC has 26 variables, identified by the letters A - Z. Variables can range in value from -32768 to + 32767, and can only take whole number values. This means that the result of an operation such as PRINT 5/7 will be zero, for example. The LET command, which is optional, is used to assign values to variables, and the CLEAR command can be used to set them all to zero. Numerical entry can be either in decimal, which is the default, or in hexadecimal by prefixing the number with "&".

1.2.2 OPERATORS

The four arithmetic operators are of course included, and the logical operators are AND, NAND, OR, NOR, EXOR. There are rules for the order of precedence of operators which the system uses to evaluate an expression, although they can always be overridden by using brackets.

In any expression, the FUNCTIONS (see below) are evaluated first. Functions from this group in the same expression are evaluated from left to right. The multiply (*) and divide (/) operators, together with the logical operators (AND, NAND, OR, NOR, EXOR) are evaluated next, again in order from left to right. The last group contains the addition (+) and subtraction (-) operators, which are evaluated last. The following examples should illustrate this:-

PRINT 16/2+9/3 is evaluated as (16/2)+(9/3) = 11

PRINT 16/2 OR 14/7 is evaluated as ((16/2) OR 14)/7 = 2

PRINT 16+3 AND 1 is evaluated as 16+(3 AND 1) = 17.

PRINT 16+3 AND 7-5 is evaluated as 16+(3 AND 7)-5 = 14

PRINT 16+3 AND 7*5 is evaluated as 16+((3 AND 7)*5) = 31

Relational operators, including =, <, <=, >, >=, <> are also included. These are often used with IF commands to make a decision within a program.

There are three other single characters that are not operators in the usual sense. These are the print formatting character ("?" or "HASH" sign), the hexadecimal display character ("@" or "AT" sign), and the hexadecimal entry character ("&"). These are often used in conjunction with the PRINT command to control data display and entry, but the & character can be used to prefix any numeric entry that you wish to be interpreted as hexadecimal. For example, typing PRINT &55 will give the result 85 - this is a quick way of converting &55 in hexadecimal to 85 in decimal. In order to convert the other way, typing PRINT @85 will give the result &55. The "?" ("HASH") character is used along with an expression to control the number of columns used to display a number - for full details please refer to the description of the PRINT command.

Note that the symbols "@" ("AT" sign) and "?" ("HASH" sign) may not be printed correctly in this manual due to limitations of the printer used. They will however display correctly on the screen.

1.2.3 FUNCTIONS

Many of the normal BASIC functions are included, including ABS, CHR, RND, SIZE, PEEK and ASC. In addition, some new functions have been added to assist in test programming. These include PASS and FAIL, which indicate the result so far of the test in progress, OPFAIL, which indicates whether the last OUTPUT operation was successful, and NOT, which simply performs a logical inversion of the expression specified. All the functions, with the exception of PASS, FAIL, SIZE and OPFAIL, have an argument which is specified in brackets.

1.2.4 DIRECT AND INDIRECT COMMANDS

There are some TESTBASIC commands that are only valid when entered directly from the keyboard and cannot be used within a program. These include RUN, NEW, SAVE, LOAD, LIST, LLIST , QUIT, EDIT and DIR. All the other commands can be executed either as part of you program or by simply typing them in. For example, the PRINT command can be used directly to perform simple calculations or hex/decimal conversion.

1.2.5 PROGRAM ENTRY

Entry of programs is similar to most other BASIC interpreters. A line number is used to provide a means of identification of any point of a program, and also to allow easy insertion, deletion and editing of lines. The EDIT command allows modifications to be made to an existing line using the cursor control keys to control editing. This eliminates the need to type in a slightly incorrect line again. We suggest that you get into the habit of entering programs with line numbers ascending by a factor of 10 each time, thereby leaving space for additional lines later if modifications are necessary.

Programs can be viewed on the screen or printer by using the LIST or LLIST commands, which can be used to view any portion of the program by entering the line numbers. The BROWSE command can also be used to display the entire program one screen at a time.

The SAVE and LOAD commands can be used to store and retrieve programs to/from the disk, respectively, and we suggest that you use the SAVE command periodically when entering a long program to avoid losing your program in the event of, for example, a power failure. The name of your SAVED program can be from 1 to 8 alphanumeric characters, but if you wish to execute test programs externally (ie. without entering PROGRAM mode) you should only use numerical filenames.

Finally the NEW command can be used to clear the entire program from memory, but make sure that it is saved on the disk prior to doing this, as it will then be gone for ever. The QUIT command will clear the program and at the same time return the system to NORMAL mode.

1.2.6 PROGRAM EXECUTION

There are two ways to execute a program that you have written, depending on whether the system is in PROGRAM mode or not. In PROGRAM mode, the RUN command causes the program in memory to be executed in ascending numerical order of line numbers, unless the program contains instructions, such as GOTO or GOSUB, that modify the normal sequence. This will be the method

you use when testing your program, since full error handling is provided. If your program "crashes", or gets stuck in a loop from which there is no exit, control can be regained by pressing the ESC key. This is quite safe and does not alter variables or your program in any way.

The second way of executing a program is called an EXTERNAL RUN operation. This can be performed without entering PROGRAM mode by simply entering a type number corresponding to the filename of your program, which must have previously been stored on the disk with an entirely numeric filename. Provided that the filename does not correspond to the number of an IC in the internal library, the system will automatically LOAD the program from the disk, run the TESTBASIC interpreter and execute the program. No error handling capability is provided, and all commands (except the SUSPEND command) that require system input/output are ignored (eg. PRINT, LPRINT, INPUT, GET, DISPLAY, CLS). After the execution of a program, the result will automatically be displayed by the system in the usual way, regardless of whether there is a DISPLAY command in the program itself.

1.2.7 ERROR HANDLING

TESTBASIC has quite comprehensive error handling, with meaningful messages to indicate the error rather than codes which must be looked up. A full list of error messages is given at the end of this manual, but most of them are self explanatory. In many cases, the error will be flagged on the screen by a question mark inserted into the offending line at the point where the error was detected. The EDIT command can then be used to make a rapid correction.

1.2.8 PROGRAM FLOW

The usual BASIC commands for directing program flow and implementing loops are included, such as GOTO to skip to a specified line number, GOSUB and RETURN to execute a sub-routine, FOR ... NEXT ... STEP to implement a repetitive loop, IF to perform conditional testing, and STOP to mark the end of your program. REM commands can be used to insert comments anywhere in your program.

1.2.9 INPUT/OUTPUT

Input and output in this context related to user input output, via the screen, keyboard or printer. The PRINT and LPRINT commands can be used to display data on the screen or printer respectively, while the INPUT and GET commands can be used to accept operator input during a program. The POKE command can be used to store data directly in memory. The CURSOR command can be used to set the cursor position on the screen. Finally, the CLS command is used to clear the screen.

1.2.10 IC TESTING COMMANDS

The following additional commands, which will be discussed in detail later, are used to perform the IC testing functions:-

```
SETUP -       to set up the IC input output pins
SUPPLY -      to define the VCC and GROUND supply pins
IN-CIRCUIT -  to define IN-CIRCUIT test environment
OUT-CIRCUIT - to define OUT-OF-CIRCUIT test environment
THRESHOLD -   to set the input threshold voltages
```


OUTPUT - to output data to the IC inputs
CLOCKH - to output a high clock pulse to an IC input
CLOCKL - to output a low clock pulse to an IC input
SENSE - to read the state of the IC outputs
CHECK - to read back data previously output to the IC
MEASURE - to measure the voltage on the IC output
COMPARE - to compare actual and expected IC outputs
DISPLAY - to display the IC result diagram

1.3 TEST PROGRAM STRUCTURE

For IC testing, most applications programs follow a similar pattern, regardless of the IC under test. One of the most important features of programming is implementing CIRCUIT COMPENSATION, which is the responsibility of you the programmer. We will now discuss the usual sequence of steps in a typical program, before going on to describe each available command in detail.

1.3.1 THE SETUP

The first step in testing an IC is to tell the system how many pins the IC under test has, and which pins are inputs and outputs. This is performed by the SETUP command, which also performs the AUTOMATIC CLIP POSITIONING operation and detects SHORTS, LINKS and FLOATING pins, although this is entirely transparent to the programmer. If required, the THRESHOLD command can be used to change the default input threshold voltages in a special application, and the IN-CIRCUIT or OUT-CIRCUIT commands can be used to tell the system where the IC under test is situated. Note that if your program is used in EXTERNAL RUN MODE the IN-CIRCUIT and OUT-CIRCUIT commands are ignored and the test environment will be determined by the mode selected by the IN/OUT key on the keypad.

1.3.2 INITIALISATION

The next step will usually be to initialise the IC under test to a known state. This involves setting the IC inputs to the correct levels to start the test, and in some cases clear or load pulses will need to be issued. The commands used here will include OUTPUT, to output data to the IC under test, and may include CLOCKH and CLOCKL, which are used to issue positive clock pulses and negative clock pulses respectively.

1.3.3 CIRCUIT COMPENSATION

One of the most difficult aspects of test programming is the CIRCUIT COMPENSATION operation. This means that your programs have to be written to take account of the way in which the IC under test is connected in the circuit.

There are two ways to approach this - if you are always testing the same IC connected in the same way, you can ignore it altogether, and write the program to test the IC as it stands. For example, if a pin on an IC was connected to ground you need not try to output a high level to it.

If however you need to test the same IC in different configurations you will wish to write your program in such a way as to automatically compensate for the circuit connections, and TESTBASIC has a two features to allow for this.

1.3.4 THE "OPFAIL" FUNCTION

The first method of circuit compensation is to use the OPFAIL function. This is a logical function returning a TRUE result if the last OUTPUT, CLOCKH or CLOCKL command was unsuccessful, and a FALSE result if the command was successfully completed. This means that you can alter your program flow using an IF OPFAIL GOTO ... command, for example, to miss out a portion of a test that you do not want to execute if the output failed. A full description of the OPFAIL function, together with examples, is given in the reference section.

1.3.5 THE "CHECK" COMMAND

The second method of compensation is to use the CHECK command to read back the result of an OUTPUT command. For example, if you want to output data to input pins of an IC, but you are not sure how they are connected, you can use the CHECK command to read back the data from the input pins and use that data for further calculation rather than the data output in the first place. In this way, whatever the connections to the input pins, your calculation will always use the same data that the IC itself sees in the circuit.

1.3.6 THE TEST OPERATION

The method of testing depends of course on the function of the IC under test, but generally consists of driving the inputs of the IC under test using the OUTPUT, CLOCKH and CLOCKL commands, calculating the expected response of the IC outputs using the built-in arithmetic and logical operations, reading the response of the outputs using the SENSE command, and comparing the two using the COMPARE command. The COMPARE command compares the actual data read from the IC with that expected, and stored the result internally so that they can subsequently be displayed.

1.3.7 DISPLAYING THE RESULTS

The results of the test can be displayed in two ways - either the PRINT command can be used in your program to display anything you want about the test, or the DISPLAY command will display the results in the form of an IC diagram in the same way that is used in normal operation.

Remember that if you are writing a program to be executed in EXTERNAL mode, PRINT and DISPLAY commands are ignored when the EXTERNAL RUN operation is performed.

1.4 SAMPLE PROGRAMS

The following programs illustrate the use of the TESTBASIC language to program three well known ics. These ICs are all included in the library, but the principles involved are identical to those you will use when writing your own programs. Program 1 is a simple program to test a 7400 QUAD NAND IC, program 2 is a more involved program to test a 74138 3-8 DECODER IC, while program 3 is a fairly complex program to test a 74163 BINARY COUNTER IC. If you refer back to the previous section, you will see that the structure of a typical program follows a pattern consisting of the following steps:-

1) SETUP - program the number of pins, the allocation of pins as inputs or outputs, automatic clip positioning, short, link and floating pin detection.

2) INITIALISATION - set up the initial state of the IC input pins at the start of test.

3) TEST - apply signals to the IC inputs and evaluate the response of the IC outputs according to the IC function.

4) DISPLAY - display the test results.

During the TEST phase of the procedure, you will usually wish to allow for hard wired connections to the IC inputs - this is AUTOMATIC CIRCUIT COMPENSATION and is probably the most difficult aspect of programming. Each of the following examples will show how this is done. Note that if you are writing a program for a custom or semi-custom IC which is used in only one configuration, then AUTOMATIC CIRCUIT COMPENSATION may not be necessary as you can test the IC as it is wired in your circuit.

We will now discuss each of the sample programs in detail. You may find it helpful to refer to a data book as you work through each example.

1.4.1 PROGRAM 1 - 7400 QUAD NAND GATE - THE SETUP

The 7400 is a QUAD NAND gate IC with 14 pins, of which pins 1,2,4,5,9,10,12,13 are inputs. It is advisable to test the 4 gates in the package individually, so that interconnections between the output of one gate and the input of another do not affect the test. The command to do this would be:-

```
10  SETUP 14 1,2
```

This tells the unit that the IC has 14 pins and that pins 1 and 2 are inputs to the IC. All remaining pins are regarded by default as outputs. This however can cause a problem in certain configurations, since the system assumes that pins 4,5,9,10,12,13 are outputs whereas they are in fact inputs to the IC. If any of these pins are connected to VCC or GROUND, this will cause a fail result since an output pin is not allowed to be connected to either supply rail. To avoid this problem, a dummy SETUP command should be used prior to the first SETUP to specify the correct input and output pins for the IC. The first two lines in the program would then be:-

```
 5  SETUP 14 1,2,4,5,9,10,12,13
10  SETUP 14 1,2
```

Note that only the first SETUP in the program checks for outputs shorted to either supply rail - the remaining SETUPS have no effect.

1.4.2 INITIALISATION

The initial states of the two inputs (pins 1 and 2) of the IC is irrelevant, since all four combinations of the two signals will be applied during the test stage to be described next.

1.4.3 THE TEST

The test in this case consists of applying signals to the IC inputs (pins 1 and 2) and evaluating the output (pin 3) according to the IC function. Firstly, consider the truth table for a NAND gate:-

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

To test this gate, all we have to do is to apply the logic signals A and B to pins 1 and 2 and check that pin 3 gives the logic signal F from the table. This could be achieved by the following program steps:-

```
20  OUTPUT 0 TO 1,2
30  SENSE F FROM 3
40  COMPARE F,1
50  OUTPUT 1 TO 1,2
60  SENSE F FROM 3
70  COMPARE F,1
80  OUTPUT 2 TO 1,2
90  SENSE F FROM 3
100 COMPARE F,1
110 OUTPUT 3 TO 1,2
120 SENSE F FROM 3
130 COMPARE F,0
```

In this program, lines 20, 50, 80 and 110 are used to output the 4 combinations of A and B from the truth table to the input pins of the gate under test. Note that the values output are in decimal (0,1,2,3) rather than binary - refer to the description of the OUTPUT command for a full explanation of this. Lines 30, 60, 90 and 120 read the binary state of the gate output into a variable (F), and lines 40, 70, 100 and 130 compare this actual value with the theoretical value from the truth table.

The above sequence uses 12 program lines to fully test the gate. If the same procedure was used to test, for example, a 4 input gate, there would be 16 combinations of the 4 inputs resulting in 48 program lines. Another way of achieving the same result in fewer program lines is to use a loop, as shown below:-

```
20  FOR X = 0 TO 3
30  OUTPUT X TO 1,2
40  SENSE F FROM 3
50  T = 1
60  IF X = 3 T = 0
70  COMPARE F,T
80  NEXT X
```

This sequence uses a variable (T) to represent the theoretical value of the output. This is usually set to 1 in line 50, but if the value of X is 3 then T is set to 0 by line 60, corresponding to the last line in the truth table shown above. The actual value (F) read from pin 3 is then compared with the theoretical value (T) in line 70. Note that by using a loop we have saved 5 program lines - for a 4 input gate the saving would

be 41 lines.

1.4.4 DISPLAY

So far there is no result displayed on the screen. There are a number of ways in which this can be achieved, depending on your application. Each COMPARE operation sets an internal flag in the event of a difference between the two values being compared, and this flag can be tested using the FAIL and/or PASS functions. Your program can then display the result directly using the PRINT command, as follows:-

```
90  IF PASS PRINT "THIS IC IS GOOD"
100 IF FAIL PRINT "THIS IC IS FAULTY"
```

Another way of displaying the result is to use the DISPLAY command, which displays the usual IC diagram and the conditions of all the pins just as if the test had been performed directly from the keypad. This would be entered as follows:-

```
90  DISPLAY
```

These are the main methods of displaying the results, but you may wish to provide a more comprehensive result display, for example displaying the entire truth table of the IC on the screen. This can be done by using PRINT statements at various points within the test loop given above.

All methods of displaying results are ignored when an IC is tested in EXTERNAL RUN mode - ie. when a TESTBASIC program is run directly from the keypad by keying in its numeric filename.

1.4.5 CIRCUIT COMPENSATION

A two input gate could be used as an inverter by tying its two input pins together, or by tying one of them to 5V. One of the most powerful features of both the DDS-40 and BoardMaster 4000 systems is to recognise this and test the gate accordingly - this is called AUTOMATIC CIRCUIT COMPENSATION. This is handled automatically by the built-in library of test software, but when programming you must provide the compensation yourself. The OPFAIL function and the CHECK command provide the means to implement circuit compensation. We will know show how this can be achieved in the program for the 7400 IC, using two methods.

1.4.6 USING OPFAIL

Your program can ignore a test if it was unable to output the specified value to the pins, using the OPFAIL function. Using the loop given above, the following example should make this clear:-

```
20  FOR X = 0 TO 3
30  OUTPUT X TO 1,2
35  IF OPFAIL GOTO 80
40  SENSE F FROM 3
50  T = 1
60  IF X = 3 T = 0
70  COMPARE F,T
80  NEXT X
```


In this example the calculation of T and the resultant COMPARE are skipped at line 35 if the OUTPUT command at line 30 failed to output the correct logic levels to pins 1 and 2. If pins 1 and 2 were tied together, the only values of X that would be correctly output would be 0 (ie. both pins low) and 3 (ie. both pins high). This means that the gate would automatically be tested as an inverter.

1.4.7 USING CHECK ... FROM

The other way is to use the actual values at the IC input pins (rather than the values output) when calculating the theoretical response of the IC, using the CHECK command. This would be implemented as follows:-

```
20  FOR X = 0 TO 3
30  OUTPUT X TO 1,2
35  CHECK Y FROM 1,2
40  SENSE F FROM 3
50  T = 1
60  IF Y = 3 T = 0
70  COMPARE F,T
80  NEXT X
```

Note the use of the CHECK command in line 35 to read back the logic levels at the inputs (pins 1 and 2) of the gate. These levels are used in line 60 to calculate the theoretical value (T) of the output. Note that after the OUTPUT command in line 30 the value of X plays no further part in the test until the next iteration of the loop. The only possible values that can exists at pins 1 and 2 are both low or both high, so the gate is again tested as an inverter.

1.4.8 SUMMARY OF 7400 PROGRAM

The above program is for 1 gate in the pack only. Repeating the above for the remaining three gates gives the following complete program for the 7400 IC:-

```
  5  SETUP 14 1,2,4,5,9,10,12,13
 10  SETUP 14 1,2
 20  FOR X = 0 TO 3
 30  OUTPUT X TO 1,2
 40  CHECK Y FROM 1,2
 50  SENSE F FROM 3
 60  T = 1
 70  IF Y = 3 T = 0
 80  COMPARE F,T
 90  NEXT X
100  SETUP 14 4,5
110  FOR X = 0 TO 3
120  OUTPUT X TO 4,5
120  CHECK Y FROM 4,5
130  SENSE F FROM 6
140  T = 1
150  IF Y = 3 T = 0
160  COMPARE F,T
170  NEXT X
180  SETUP 14 9,10
190  FOR X = 0 TO 3
```



```
200 OUTPUT X TO 9,10
210 CHECK Y FROM 9,10
220 SENSE F FROM 8
230 T = 1
240 IF Y = 3 T = 0
250 COMPARE F,T
260 NEXT X
270 SETUP 14 13,12
280 FOR X = 0 TO 3
290 OUTPUT X TO 13,12
300 CHECK Y FROM 13,12
310 SENSE F FROM 11
320 T = 1
330 IF Y = 3 T = 0
340 COMPARE F,T
350 NEXT X
360 DISPLAY
```

The above program is quite long, but it can be reduced by using a subroutine to perform the test for each gate. In this example, the pin numbers of the gate to be tested are passed to the subroutine in the variables A, B and C. The subroutine then performs the test as derived previously. The complete program is shown below:-

```
  5  SETUP 14 1,2,4,5,9,10,12,13
 10  SETUP 14 1,2
 20  A = 1: B = 2: C = 3: GOSUB 500
 30  SETUP 14 4,5
 40  A = 4: B = 5: C = 6: GOSUB 500
 50  SETUP 14 9,10
 60  A = 9: B = 10: C = 8: GOSUB 500
 70  SETUP 14 13,12
 80  A = 13: B = 12: C = 11: GOSUB 500
 90  DISPLAY
100  STOP
500  FOR X = 0 TO 3
510  OUTPUT X TO A,B
520  CHECK Y FROM A,B
530  SENSE F FROM C
540  T = 1
550  IF Y = 3 T = 0
560  COMPARE F,T
570  NEXT X
580  RETURN
```

1.4.9 PROGRAM 2 - 74138 3 TO 8 LINE DECODER

This IC decodes three input signals and outputs an active low signal on one of 8 output pins, the other 7 outputs remaining high. It also has one active high and two active low enable pins. In this case we will give the complete program first, followed by the explanation:-

```
 10  SETUP 16 1,2,3,4,5,6
 20  T = &FF
 30  OUTPUT &28 TO 1,2,3,4,5,6
 40  GOSUB 500
 50  OUTPUT &30 TO 1,2,3,4,5,6
```



```
60  GOSUB 500
70  OUTPUT 0 TO 1,2,3,4,5,6
80  GOSUB 500
90  OUTPUT &20 TO 1,2,3,4,5,6
100 T = &FE
110 FOR X = 0 TO 7
120 OUTPUT X TO 1,2,3
130 GOSUB 500
140 T = ((T * 2) OR 1) AND &FF
150 NEXT X
160 DISPLAY
170 STOP
500 IF OPFAIL RETURN
510 SENSE A FROM 15,14,13,12,11,10,9,7
520 COMPARE A,T
530 RETURN
```

In this program, line 10 sets up the input and output pins in the usual way. Line 20 initialises the theoretical value of the 8 outputs to all high (hexadecimal FF), and at line 30 the IC is disabled by driving pins 4,6 high and pins 1,2,3,5 low. In the disabled state the outputs should all be high and this is checked at line 40 using the subroutine at line 500 which reads the 8 outputs and compares them with the theoretical value T. Note that the subroutine will return without performing any function if the previous output command was unsuccessful due to circuit conditions - this provides AUTOMATIC CIRCUIT COMPENSATION for the program. The disable function is tested twice more in lines 50 to 80 using pins 5 and 6 to disable the IC.

At line 90, the IC inputs are configured to enable the IC, and the first expected value (T) is initialised to hexadecimal FE which is equivalent to pin 15 low and all other outputs high. A loop follows consisting of lines 110 to 150, which outputs all 8 combinations of the three input bits to be decoded (pins 1,2 and 3) and tests for the correct output pattern. Line 140 effectively shifts the value of T one place to the left, filling in a high level at the least significant bit position and truncating the new value to 8 bits. This corresponds to pin 14 low and all other outputs high. This process is repeated for all 8 values of the encoded inputs, after which the results are displayed and the test is complete.

1.4.10 PROGRAM 3 - 74163 BINARY COUNTER

This IC is a 4 bit synchronous counter with clear, clock, load, data and inhibit inputs, and 4 Q outputs along with a ripple carry. The following program is only one of many ways in which the IC could be tested:-

```
10  SETUP 16 1,2,3,4,5,6,7,9,10
20  OUTPUT &13 TO 1,2,7,10,9
24  REM
25  REM *** CHECK LOAD FUNCTION WITH 0101 ***
26  REM
30  OUTPUT &5 TO 3,4,5,6
40  OUTPUT 0 TO 9
50  IF OPFAIL GOTO 100
60  CLOCKL TO 2
70  OUTPUT 1 TO 9
80  CHECK T FROM 3,4,5,6
```



```
90  GOSUB 500
100 OUTPUT 1 TO 9
104 REM
105 REM *** CHECK LOAD FUNCTION WITH 1010 ***
106 REM
110 OUTPUT &A TO 3,4,5,6
120 OUTPUT 0 TO 9
130 IF OPFAIL GOTO 180
140 CLOCKL TO 2
150 OUTPUT 1 TO 9
160 CHECK T FROM 3,4,5,6
170 GOSUB 500
180 OUTPUT 1 TO 9
184 REM
185 REM *** CHECK CLEAR FUNCTION ***
186 REM
190 OUTPUT 0 TO 1
200 IF OPFAIL GOTO 250
210 CLOCKL TO 2
220 OUTPUT 1 TO 1
230 T = 0
240 GOSUB 500
250 OUTPUT 1 TO 1
254 REM
255 REM *** CHECK COUNT DISABLE FUNCTION ***
256 REM
260 SENSE T FROM 14,13,12,11
270 OUTPUT 0 TO 7,10
280 IF OPFAIL GOTO 320
290 CLOCKL TO 2
300 OUTPUT 3 TO 7,10
310 GOSUB 500
320 OUTPUT 3 TO 7,10
324 REM
325 REM *** CHECK COUNT FUNCTION ***
326 REM
330 SENSE T FROM 14,13,12,11
340 FOR X = 0 TO 16
350 GOSUB 500
360 CLOCKL TO 2
370 T = T + 1
380 IF T = 16 T = 0
390 NEXT X
400 DISPLAY
410 STOP
414 REM
415 REM *** SUBROUTINE TO CHECK OUTPUTS ***
416 REM
500 SENSE A FROM 14,13,12,11
510 COMPARE A,T
520 R = 0
530 IF A = 15 R = 1
540 SENSE B FROM 15
550 COMPARE B,R
560 RETURN
```


This program is in several sections, testing one function of the IC at a time. The load function is tested in 30 to 160, first by loading the 4 data inputs on the counter with 0101 (binary), then by loading with 1010. This means that each data input pin is tested in both high and low states. Notice that the theoretical value of the counter outputs after the load operation (T) is calculated by using the CHECK command to read the data inputs, thus providing AUTOMATIC CIRCUIT COMPENSATION. A subroutines located at line 500 is used to read the counter outputs and compare them with the expected value given in T. The subroutine also checks the correct state of the ripple carry output (pin 15). Note at lines 50 and 130 that the load test is skipped if the tester is unable to pull the load input (pin 9) low (ie. if the load input is tied to 5V).

The clear function is tested next in lines 190 to 250. Again, if the clear input (pin 1) is tied high than no testing takes place and this section of the program is skipped.

The count disable function is tested next by trying to clock the counter while both its enable pins are driven low. The counter outputs should not change during this process. Note that the initial state of the counter outputs is read in line 260 since the program has no way of knowing whether the load and/or clear tests were carried out (due to circuit connections) and therefore it has no way of knowing the current state of the counter. As before, if the test system cannot pull both pins 7 and 10 low, it will drive them back high and skip this section of the test.

The final section of the test is to check the count function. This is a loop starting at line 330 with a SENSE command to read the current state of the counter outputs. Sixteen clock pulses follow, with the value of T being incremented each time to keep pace with the counter outputs. Note at line 380 the value of T is reset to zero when it reaches 16, just as the 4 bit counter itself would behave.

Finally, at line 400, the results are displayed and the program terminates.

1.4.11 DEBUGGING YOUR PROGRAMS

Most programmers are aware that programs do not in general work first time. To debug the program, the best method is to insert DISPLAY and STOP commands at convenient points to allow you to track down the point at which an error occurs. The PRINT or LPRINT commands can also be used to display the value of variables during the execution of a program. If you were to insert LPRINT commands to print the actual and theoretical values prior to each COMPARE command, for example, you would get a list of them on the printer. This would enable you to easily spot the occurrence of an error.

If you are writing a program to test an IC in-circuit, you may find it convenient to test and debug the program with an IC out of circuit in the first instance. The OUT-CIRCUIT command, which should be the first line in your program, can be used to allow this. The IC under test can then be inserted into the ZIF socket.

Remember that screen/printer output and keyboard input commands such as PRINT, LPRINT, INPUT, GET are ignored in EXTERNAL RUN mode, so you can leave any debugging commands in your program without corrupting the screen

display. IN-CIRCUIT and OUT-CIRCUIT commands are also ignored in EXTERNAL RUN mode.

Finally, do not forget to periodically SAVE the program you are writing on the disk to minimise loss in the event of a power failure or accidental switch off. This applies particularly when writing long and complex programs which may result in many hours of work lost if the unit were to be accidentally powered down.

1.5 DETAILED COMMAND AND FUNCTION DESCRIPTION

This section is a full description of all the commands and functions that TESTBASIC understands, together with examples of how they are used in a program. The syntax is quite precise, but you can insert as many spaces as you like between commands and operators to increase the readability of your programs.

In the command descriptions, the following terms are used:-

Variable - a 16 bit 2s complement integer identified by a single letter from A to Z.

Expression - a mathematical and/or logical expression made up of functions, operators, variables and constants

Character - an single alphanumeric character from the keyboard

String - a list of characters enclosed by single or double quotation marks.

[] - square brackets indicates that the item inside the brackets is optional.

Pinlist(n) - a pin list is a list of expressions corresponding to the pins of the IC under test, separated by commas. The maximum number of pins in the list is given in brackets, and is usually 8 except where stated. Note that each pin number can if required be an expression, although constants are more usual.

The rest of this section is an alphabetical list of all the commands and functions.

ABS The ABS function converts negative numbers into their equivalent positive value but leaves positive numbers as they are. For example the absolute value of -6 is 6 and the absolute value of 2 is 2.

The ABS function can be used to calculate the difference between two variables when you don't know which is the greater of the two.

Program examples:
```
150  J = ABS(Q)
300  IF ABS(A-B) > 5 GOTO 250
```

Syntax: ABS(expression)

AND The AND operator is used to perform a bitwise logical AND operation between two numerical expressions. Note that the AND operator cannot be used as a relative operator, ie. the command IF A > 3 AND B > 5 GOTO 50 is illegal.

For example, if variable A = 54 and variable B = 21 the operation C = A AND B would proceed as follows:-

```
A = 54 decimal  00110110 binary  &36 hexadecimal
B = 21 decimal  00010101 binary  &15 hexadecimal
C = 20 decimal  00010100 binary  &14 hexadecimal
```

Program examples:
```
300 S=A AND G
400 PRINT 3 AND 5
600 OUTPUT A AND &FE TO 1,2,3,5,9
```

Syntax: expression AND expression

See also: NAND, OR, NOR, EXOR, NOT

ASC The ASC function produces the ASCII value of the character given in the argument. This is complementary to the CHR function.

Program examples:
```
120 X = ASC(H)
180 PRINT ASC(E)
```

Syntax: ASC(character)

See also: CHR

BROWSE The BROWSE command displays your program 20 lines at a time (approximately one screen full). Pressing any key allows the next screen of program to be displayed. Pressing CONTROL-C (the CONTROL and C keys together) will abort the BROWSE command altogether. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

Syntax: BROWSE

See also: LIST, LLIST

CHECK ... FROM The CHECK command reads the logic states (HIGH or LOW) at the pins of the IC under test into a variable, where the least significant bit of the variable corresponds to the first pin number given in the list after the CHECK command. Those bits of the variable with no corresponding pin number given in the command will be set to zero. There is a minimum of one and a maximum of 8 pin numbers allowed with this command.

The CHECK command is normally used to read back the logic states of the INPUTS of the IC under test, to ensure that they have assumed the correct levels after a previous OUTPUT command. Any differences (due to hard wired circuit connections) between the actual values output to the IC under test and those read back from it can then be discovered. The program can then take these differences into account, using the actual values read using the CHECK command for subsequent calculations rather than the values output. This is the principle way of effecting the AUTOMATIC CIRCUIT

COMPENSATION function in your programs. The command is similar to the SENSE command except that there is no MID LEVEL check performed.

Program example:

```
10  ...
20  FOR A = 0 TO 3
30  OUTPUT A TO 2,3
40  CHECK B FROM 2,3
50  IF B = 0 J = 1
60  IF B >= 1 J = 0
70  SENSE X FROM 1
80  COMPARE X,J
90  NEXT A
100 .....
```

This example shows part of a program to test a 7402 quad 2 input NOR gate. Note how at line 40 the state of pins 2 and 3 after the OUTPUT operation at line 30 is CHECKED. The value of B will then reflect the actual state of pins 2 and 3 in any circuit configuration, allowing the remainder of the program to properly calculated the expected output response in lines 50 and 60.

Syntax: CHECK variable FROM pinlist(8)

See also: SENSE, OPFAIL

CHR The CHR function produces a character from the ASCII code given in the argument. This is complementary to the ASC function.

Program example:

```
10 A = CHR(55)
50 PRINT CHR(41)
```

Syntax: CHR(expression)

See also: ASC

CLEAR The CLEAR command sets all TESTBASIC variables to zero.

Program example: 500 CLEAR

Syntax: CLEAR

CLOCKH TO ... , CLOCKL TO ...

The CLOCKH command outputs the sequence HIGH, LOW to each pin in the list following the CLOCKH command. The CLOCKL command outputs the sequence LOW, HIGH to each pin in the list following the CLOCKL command. In either case, the value of OPFAIL will be set to TRUE if the correct output sequence was not executed because of the circuit configuration.

These commands will often be used to issue clear, clock, load pulses avoiding the need for two successive OUTPUT commands.

Program examples:

```
10 CLOCKH TO 1,B
90 CLOCKL TO A,A + 2
```

Syntax:

```
CLOCKH TO pinlist(8)
CLOCKL TO pinlist(8)
```


CLS The CLS command clears the screen and returns the cursor to its home position (top left hand corner).

This command is ignored in an EXTERNAL RUN operation.

Program example: 500 CLS

Syntax: CLS

COMPARE The COMPARE command is used to compare the actual output of the IC under test with that calculated by your program. If a difference exists the PASS function is set to FALSE and the FAIL function is set to TRUE. In addition, the differences are stored internally where they are available for use by the DISPLAY command. If the values are identical, the PASS and FAIL functions are unchanged.

Program example:
```
50 OUTPUT N TO 1,4,5
60 T=0
70 SENSE A FROM 2
80 COMPARE A,T
90 ....
```

Syntax: COMPARE actual expression, theoretical expression

CURSOR The CURSOR command is used to set the cursor position on the screen, to facilitate screen formatting of tables etc. The row and column numbers of the desired cursor location can be specified, starting at row 0 column 0 at the top left up to row 23 column 79 at the bottom right. Note that the row and column numbers may be expressions, as in the following example.

Program examples:
```
20 CURSOR X,Y
60 CURSOR A+3,B/5
```

Syntax: CURSOR row expression, column expression

DIR The DIR command displays the directory of the disk in the same format as that used by the DIRECTORY function in normal system operation. Pressing any key after the DIR command will revert to TESTBASIC operation. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

Syntax: DIR

DISPLAY The DISPLAY command displays the stored results of all the previous COMPARE commands as an IC diagram on the screen. It is often the last command in a program.

This command is ignored in an EXTERNAL RUN operation.

Program example: 500 DISPLAY

Syntax: DISPLAY

EDIT Typing EDIT followed by a line number gives access to the line editing facility. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

The following editing keys are available:-

RIGHT cursor key or CONTROL-D: move the cursor right by one character position.

LEFT cursor key or CONTROL-S: move the cursor left by one character position.

DEL key: delete the character to left of the cursor.

CONTROL-G: delete the character under the cursor.

CONTROL-V: select INSERT or OVERWRITE modes. In INSERT mode, characters typed are inserted into the line without deleting those already present, whilst in OVERWRITE mode the characters present are replaced. The current status is displayed on the bottom line of the screen.

A Carriage return enters the edited line and exits from the line editor.

Syntax: EDIT line number

EXOR The EXOR operator is used to perform a bitwise logical exclusive OR operation between two numerical expressions. Note that the EXOR operator cannot be used as a relative operator, ie. the command IF A > 3 EXOR B > 5 GOTO 50 is illegal.

For example, if variable A = 54 and variable B = 21 the operation C = A EXOR B would proceed as follows:-

```
A = 54 decimal  00110110 binary  &36 hexadecimal
B = 21 decimal  00010101 binary  &15 hexadecimal
C = 35 decimal  00100011 binary  &23 hexadecimal
```

Program examples:

```
200 S = A EXOR B
400 PRINT 3 EXOR 5
600 OUTPUT A EXOR 1 TO 1,2,3,5,9
```

Syntax: expression EXOR expression

See also: AND, NAND, OR, NOR, NOT

FAIL The FAIL function is used to indicate the outcome of COMPARE commands throughout your program and hence the result of the test. It is automatically initialised to FALSE when running your program, and it will be set to TRUE by any COMPARE command that indicated a difference between actual and theoretical values. Note that only one bad COMPARE command is needed to set FAIL to TRUE, even if subsequent COMPARE commands are all good.

Program example:

```
70 IF FAIL GOTO 100
90 IF FAIL PRINT "THIS IC IS BAD"
```

Syntax: FAIL See also: PASS

FOR ... STEP The FOR command is used in conjunction with the NEXT command and optionally with the STEP command to implement a loop whereby a series of instructions can be repeated a specified number of times. The variable specified in the FOR command is initialised to the value of the expression given in the FOR command, and execution then continues. When a NEXT command referring to the same variable is encountered, the value that was specified in the STEP command, if present, is added to the variable. If no STEP command was present a value of 1 is used. If the result of this addition is less than the maximum value given in the FOR command, execution then continues at the line after the FOR command, otherwise execution continues at the next line after the NEXT command.

Program example:
```
10 FOR A = 1 TO 8 STEP 2
20 PRINT A
30 NEXT A
```

The above program will print out the numbers 1,3,5,7

Syntax: FOR variable = expression TO expression [STEP expression]

See also: NEXT

GET The GET command is used to get a key from the keyboard. It returns an ASCII code corresponding to the key pressed. If no key is pressed, the system will wait until a key is pressed.

This command is ignored in an EXTERNAL RUN operation.

Program example:
```
60 GET X
```

Syntax: GET variable See also: INPUT

GOSUB The GOSUB command is used to transfer execution to a specified line number, where execution continues until a RETURN command is encountered. The RETURN command causes execution to continue at the line following the original GOSUB command.

This command is used when a section of your program needs to be used in a number of different places within a program. This section can be made into a SUBROUTINE which can then be called from anywhere within the main program. This is done by the GOSUB instruction followed by the line number of the SUBROUTINE. The RETURN command is placed at the end of the subroutine which returns the program to the line after the GOSUB which called it.

Program example:
```
200 GOSUB 5000
210 ....

5000 PRINT "THIS IS A SUBROUTINE"
5010 RETURN
```

Syntax: GOSUB expression

See also: RETURN

GOTO This instruction makes the program jump to a specified line number given after the GOTO instruction instead of continuing to the next line on in the program.

Program example: 20 GOTO 170

Syntax: GOTO expression

IF The IF command is used to test a condition which can then be used to control the subsequent flow of the program. It uses one of the RELATIVE operators which are given below, and it can also be used with the PASS, FAIL and OPFAIL functions. The condition following the IF command is evaluated, and if the result is TRUE the command following the condition is executed, otherwise it is ignored.

The relative operators and their meanings is as follows:-

```
=    is equal to
<    is less than
<=   is less than or equal to
>    is greater than
>=   is greater than or equal to
<>   is not equal to
```

Program examples:
```
10 IF A = 5 GOTO 300
50 IF Q > 8 PRINT "FAIL"
80 IF OPFAIL GOSUB 300
90 IF J = 1 M = 3
```
Syntax: IF expression rel. operator expression command

IN-CIRCUIT The IN-CIRCUIT command is used to inform the test system that the IC under test is to be tested IN-CIRCUIT. This means that the input/output signals will be directed to/from the 40 way test lead during the test rather than to the ZIF socket.

This command is ignored in an EXTERNAL RUN operation.

Program example: 30 IN-CIRCUIT
Syntax: IN-CIRCUIT

See also: OUT-CIRCUIT

INPUT The INPUT command is used to enter values into your program so that they can be used in calculations or test routines. If a character string enclosed in quote marks is found after the INPUT command, it will be displayed on the screen as a prompt. Any number of variables can be initialised with one INPUT command by separating them with commas.

This command is ignored in an EXTERNAL RUN operation.

Program examples:
```
10 INPUT X
50 INPUT "ENTER LOW THRESHOLD"T
40 INPUT "ENTER HIGH THRESHOLD",H
70 INPUT "ENTER HIGH & LOW THRESHOLDS",H,L
```
Syntax: INPUT [string]variable,variable... See also: GET

LET The LET command is used to assign a number or expression to a variable. It is an optional command and may be omitted.

Program examples:
```
10 LET A = 10
50 LET B = C - G
60 A = 4
```

Syntax: [LET] variable=expression

LIST, LLIST The LIST command is used to display your program on the screen, whilst the LLIST command will display it on the printer. These commands are DIRECT commands, which means that they can only be executed from the keyboard and not from within a program.

Examples:
```
LIST       lists the entire program
LIST 10    lists only line number 10
LIST 10-20 lists lines from 10 to 20 inclusive
LIST -20   lists lines up to line 20 inclusive
LIST 10-   lists lines from line 10 onwards
```

Syntax:
```
LIST [line number][-][line number]
LLIST [line number][-][line number]
```

See also: BROWSE

LOAD The LOAD command is used to load a program from a floppy disk into memory. The command is followed by the file name of the program which may be up to 8 characters long. Any program already present in memory is deleted. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

Program example: LOAD TESTPROG

Syntax: LOAD filename

See also: SAVE

LPRINT See PRINT

MEASURE The MEASURE command is used to measure the voltage of any pin on the IC under test. The value read from the pin is placed in the specified variable. The range of possible values extends from -10V giving a value of -10000 to +9.99 volts giving a value of 9999.

Program example: MEASURE A FROM 1

Syntax: MEASURE variable FROM pinlist(1)

NAND The NAND operator is used to perform a bitwise logical NAND (NOT AND) operation between two numerical expressions. Note that the NAND operator cannot be used as a relative operator, ie. the command IF A > 3 NAND B > 5 GOTO 50 is illegal.

For example, if variable A = 54 and variable B = 21 the operation C = A NAND B would proceed as follows:-


```
A = 54 decimal  00110110 binary  &36 hexadecimal
B = 21 decimal  00010101 binary  &15 hexadecimal
C = 235 decimal 11101011 binary  &EB hexadecimal
```

Program examples:
```
200 S = A NAND B
400 PRINT 3 NAND 5
600 OUTPUT A NAND 1 TO 1,2,3,5,9
```

Syntax: expression NAND expression

See also: AND, OR, NOR, EXOR, NOT

NEW The NEW command is used to remove a previous program from memory before entering a new program. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

Syntax: NEW

NEXT The NEXT command is used in conjunction with the FOR ... STEP command to implement a loop whereby a series of instructions can be repeated a specified number of times. The variable specified in the FOR command is initialised to the value of the expression given in the FOR command, and execution then continues. When a NEXT command referring to the same variable is encountered, the value that was specified in the STEP command, if present, is added to the variable. If no STEP command was present a value of 1 is used. If the result of this addition is less than the maximum value given in the FOR command, execution then continues at the line after the FOR command, otherwise execution continues at the next line after the NEXT command.

Program example:
```
10 FOR A = 1 TO 8 STEP 2
20 PRINT A
30 NEXT A
40 ....
```

The above program will print out the numbers 1,3,5,7

Syntax: NEXT variable

See also: FOR ... STEP

NOR The NOR operator is used to perform a bitwise logical NOR (NOT OR) operation between two numerical expressions. Note that the NOR operator cannot be used as a relative operator, ie. the command IF A > 3 NOR B > 5 GOTO 50 is illegal.

For example, if variable A = 54 and variable B = 21 the operation C = A OR B would proceed as follows:-

```
A = 54 decimal  00110110 binary  &36 hexadecimal
B = 21 decimal  00010101 binary  &15 hexadecimal
C = 200 decimal 11001000 binary  &C8 hexadecimal
```

Program examples:
```
200 S = A NOR B
400 PRINT 3 NOR 5
600 OUTPUT A NOR 1 TO 1,2,3,5,9
```

Syntax: expression NOR expression

See also: AND, NAND, OR, EXOR, NOT

NOT The NOT function is used to logically invert the specified expression.

Program examples:
```
10 IF NOT(A) = 5 GOTO 150
50 A = NOT(B*C)
```

Syntax: NOT(expression)

See also: AND, NAND, OR, NOR, EXOR

OPFAIL The OPFAIL function is used to provide a means of implementing AUTOMATIC CIRCUIT COMPENSATION within your program. If the last OUTPUT, CLOCKH or CLOCKL command was unsuccessful because of circuit connections, the value of OPFAIL will be set to TRUE, otherwise it will be set to FALSE. It can then be tested with an IF command.

Program example:
```
30 IF OPFAIL GOTO 200
```

Syntax: OPFAIL

See also: CHECK

OR The OR operator is used to perform a bitwise logical OR operation between two numerical expressions. Note that the OR operator cannot be used as a relative operator, ie. the command IF A > 3 OR B > 5 GOTO 50 is illegal.

For example OR 1 TO 1,2,3,5,9

Syntax: expression OR expression

See also: AND, NAND, NOR, EXOR, NOT

OUT-CIRCUIT The OUT-CIRCUIT command is used to inform the test system that the IC under test is to be tested OUT-OF-CIRCUIT. This means that the input/output signals will be directed to/from the ZIF socket during the test rather than to the test lead.

This command is ignored in an EXTERNAL RUN operation.

Program example:
```
30 OUT-CIRCUIT
```

Syntax: OUT-CIRCUIT

See also: IN-CIRCUIT

OUTPUT The OUTPUT command is used to output data to the input pins of the IC under test. The least significant bit of the data is output to the first pin in the list following the OUTPUT command, followed by the next bit to the next pin until there are no more pins given. If the desired data was not successfully output due to circuit connections, the OPFAIL function will be set to TRUE, otherwise it will be FALSE. A maximum of 8 pins can be controlled with one OUTPUT command. Note that the data to be output and the pin numbers can be expressions if required.

Program examples:
```
40 OUTPUT A TO 1,2,5
80 OUTPUT B + 2 TO X, X + 1, X + 2
```

Syntax: OUTPUT expression TO pinlist(8)

PASS The PASS function is used to indicate the outcome of COMPARE commands throughout your program and hence the result of the test. It is automatically initialised to TRUE when running your program, and it will be set to FALSE by any COMPARE command that indicated a difference between actual and theoretical values. Note that only one bad COMPARE command is needed to set PASS to FALSE, even if subsequent COMPARE commands are all good.

Program examples:
```
70 IF PASS GOTO 100
90 IF PASS PRINT "THIS IC IS GOOD"
```

Syntax: PASS

See also: FAIL

PEEK The PEEK function is used to obtain the contents of a specified 8 bit memory location. Note that if you attempt to PEEK an illegal address within the system, an error will occur. The upper limit of the allowable address range is &EFFF in hexadecimal, but the lower limit depends on the size of program entered. With no program present the lower limit is around &B080 hexadecimal.

Program examples:
```
20 PRINT PEEK(A)
50 B = PEEK(X+4)
```

Syntax: PEEK (expression)

See also: POKE

POKE The POKE command is used to store 8 bit data directly in memory. The first expression following the POKE command is the address, and the second expression is the data to be stored at that address. Note that if you attempt to POKE an illegal address within the system, an error will occur. The upper limit of the allowable address range is &EFFF in hexadecimal, but the lower limit depends on the size of program entered. With no program present the lower limit is around &B080 hexadecimal.

Program examples:
```
30 POKE A,B
40 POKE X/34,25
```

Syntax: POKE expression,expression

See also: PEEK

PRINT, LPRINT The PRINT command is used to display information on the screen, and the LPRINT command to display information on the printer. There are several forms of the print command, which we will discuss in turn using examples.

These commands will be ignored in an EXTERNAL RUN operation.

Program examples:

```
20 PRINT A,B,C
30 PRINT
40 PRINT "THE VALUE OF A IS ",A
50 PRINT A*4
60 PRINT ?4,A,?8,B,?X,C
70 PRINT A,@A
```

In line 20, the values of A, B and C are printed on the same line, with each number occupying 6 character positions. Leading zeros in the number are replaced by spaces.

In line 30, a carriage return followed by a line feed is printed, ie. a blank line is printed

In line 40 the string is printed first, followed by the value of A on the same line.

In line 50 the result of the calculation "A times 4" is printed.

In line 60, the FORMAT command (? or "HASH" sign) is used to control the number of character positions occupied by each number printed. In this example, A will occupy 4 positions, B will occupy 8 and C will occupy the number of character positions given by the value of X. If no FORMAT command is used, a value of 6 is assumed. The FORMAT command is only effective for the line containing it.

In line 70, the value of A is printed first in decimal and then in hexadecimal.

Note that the symbols "@" ("AT" sign) and "?" ("HASH" sign) may not be printed correctly in this manual due to limitations of the printer used. They will however display correctly on the screen.

Syntax: PRINT [expression][?expression][string],.....

QUIT The QUIT command is used to return to normal test system operation, starting in NORMAL mode. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

Syntax: QUIT

REM The REM command is used to insert comments in your program to explain the operation of the program. REM commands are ignored by the TESTBASIC interpreter.

Program example:

```
10 REM SETUP I/O PINS
20 SETUP 14 1,2,4,5
30 REM SET THRESHOLD VOLTAGES
40 THRESHOLD 80,240,140
```



```
50 REM CLEAR LATCH
60 OUTPUT 1 TO 3
70 CLOCKL TO 3
```

Syntax: REM

RND The RND function is used to obtain a pseudo-random number from zero up to the value specified in the RND function. The argument of the RND function must be a positive expression.

Program examples:
```
20 PRINT RND(10)
30 OUTPUT RND(&1F) TO 1,2,3,7
```

Syntax: RND (positive expression)

RUN The RUN command is used to execute the program stored in memory, starting at the lowest line number present. The screen is cleared, and the internal functions PASS, FAIL and OPFAIL are set to TRUE, FALSE and FALSE respectively. In addition, the internal table used to store the results of COMPARE commands is cleared. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

Syntax: RUN

SAVE The SAVE command is used to store a program from memory onto a floppy disk. The command is followed by the file name of the program which may be up to 8 characters long. If a file with this name already exists on the disk, you are given the option of overwriting it. This command is a DIRECT command, which means that it can only be executed from the keyboard and not from within a program.

Program example: SAVE TESTPROG

Syntax: SAVE filename

See also: LOAD

SENSE ... FROM The SENSE command reads the logic states (HIGH or LOW) at the pins of the IC under test into a variable, where the least significant bit of the variable corresponds to the first pin number given in the list after the SENSE command. Those bits of the variable with no corresponding pin number given in the command will be set to zero. There is a minimum of one and a maximum of 8 pin numbers allowed with this command.

The SENSE command is normally used to read the logic states of the OUTPUTS of the IC under test, prior to comparing them with the expected response which is calculated in your program. The SENSE command is similar to the CHECK command, except that the MID LEVEL test is performed as each pin is read.

Program example:
```
10  ...
20  FOR A = 0 TO 3
30  OUTPUT A TO 2,3
40  CHECK B FROM 2,3
50  IF B = 0 J = 1
```



```
60  IF B >= 1 J = 0
70  SENSE X FROM 1
80  COMPARE X,J
90  NEXT A
```

This is the same example as used for the CHECK command previously. At line 70 the state of pin 1 is read into the variable X, before being compared with the expected value which was calculated in J by lines 50 and 60.

Syntax: SENSE variable FROM pinlist(8)

See also: CHECK

SETUP The SETUP command is used at the start of every program to perform four functions:-

1) To tell TESTBASIC the number of pins of the IC under test.

2) To tell TESTBASIC which pins of the IC under test are inputs and which are outputs.

3) To perform the AUTOMATIC CLIP POSITIONING operation.

4) To detect and store any LINKS, SHORTS, FLOATING or OPEN CIRCUIT conditions.

Functions 3 and 4 are transparent to the programmer, who only needs to enter details of functions 1 and 2. The SETUP command has 2 arguments, the first being the number of pins of the IC under test and the second being a list of pins that are inputs to the IC under test. There must always be at least 1 pin defined as an input.

Program example 1: 10 SETUP 20 1,2,5,12,15,19

In this SETUP command, we are about to test a 20 pin IC, of which pins 1,2,5,12,15 and 19 are inputs.

Program example 2: 10 SETUP 0 1,2,3

In this modified form of the SETUP command, no power supply pin check is performed and pins 1,2,3 of the 40 way clip are defined as outputs from the test system. This form of the command is useful when functionally testing modules or small PCB's.

Syntax: SETUP expression pinlist(40)

SIZE The SIZE function is used to obtain the amount of available memory left on the system.

Program examples:
```
20 PRINT SIZE
40 X = SIZE/2
```

Syntax: SIZE

STOP The STOP command is used to cause execution of the program to cease.

Program example: 30 STOP Syntax: STOP

SUPPLY The SUPPLY command is used to define the VCC and Ground pin numbers for the IC under test, if they are not in the normal corner positions. Note that if the SUPPLY command is used, the AUTOMATIC CLIP POSITIONING feature is disabled and the clip must be applied to the IC with pin 1 of the clip on pin 1 of the IC. The SUPPLY command has two arguments, the first being the pin number of the VCC pin of the IC under test and the second being the pin number of the GROUND pin.
Program example: 30 SUPPLY 12,3

This command defines pin 12 of the IC under test as the VCC pin, and pin 3 as the ground pin.

Syntax: SUPPLY expression,expression

SUSPEND The SUSPEND command is used to halt a program until any key is pressed, thereby allowing the user to make a measurement or some other function. This is the only command that requires input that is available in EXTERNAL RUN mode.

Program example: 20 SUSPEND

Syntax: SUSPEND

THRESHOLD The optional THRESHOLD command is used to alter the values of threshold voltages used by the system when evaluating the logic state (HIGH, LOW, MID LOW or MID HIGH) of a pin. There are 3 threshold voltages - LOW, HIGH and SWITCHING. Voltages below the LOW threshold are regarded as LOW, voltages above the HIGH threshold are regarded as HIGH, while voltages in between LOW and SWITCHING are regarded as MID LOW, and voltages in between SWITCHING and HIGH are regarded as MID HIGH. The default values are 0.5 volts for the LOW threshold, 2.4 volts for the HIGH threshold and 1.2 volts for the switching threshold. These can be changed using the CONFIGURE operation and the voltages set by the CONFIGURE operation will be used if no THRESHOLD command is specified in your program. Using the THRESHOLD command will override the CONFIGURE operation, but only for the duration of the program being executed.

The THRESHOLD command has 3 arguments, which are the LOW, HIGH and SWITCHING threshold voltages respectively. Each voltage is defined by an expression evaluating to a number in the range zero (0 volts) to 497 (4.97 volts).

Program example: 20 THRESHOLD 40,240,140

In this example the threshold voltages are set to 0.4 volts, 2,4 volts and 1.4 volts respectively.

Syntax: THRESHOLD expression, expression, expression

1.6 ERROR MESSAGE DESCRIPTION

The following is a list of the system error messages, together with an explanation of possible causes of the error:-

SYNTAX ERROR - This is caused by all manner of errors, such as missing commas or spaces. Check carefully the entered line, looking for a question mark which indicates the position of the error.

RETURN WITHOUT GOSUB - A RETURN command was found without a previous GOSUB.

NEXT WITHOUT FOR - A NEXT command was found without a previous FOR.

RESULT OVERFLOW - If the result of a calculation is outside the range of -32768 to +32767, this error occurs.

MUST BE POSITIVE - The argument of a random number (RND) function must be positive.

NO SUCH LINE - A GOTO, GOSUB or EDIT command referenced a line number that does not exist.

DIVIDE BY ZERO - An attempt was made to divide a number by zero.

NO MORE MEMORY - During entry of a program, or during execution of a GOSUB or FOR ... NEXT command, the system ran out of memory to perform the operation.

ILLEGAL POKE ADDRESS - An attempt was made to POKE to an address outside the legal range.

ILLEGAL PEEK ADDRESS - An attempt was made to read data from an illegal address.

PIN NUMBER TOO LARGE - During the SETUP command, an attempt was made to set up an IC with more than 40 pins. During other IC testing commands, an attempt was made to reference a pin number greater than that specified in the last SETUP command.

TOO MANY PINS - More than 8 pins was specified in the pin list for an IC testing command.

DATA TOO LARGE FOR PINS - The data is too large to "fit" onto the specified of pins in a 1 bit per pin binary basis.

THRESHOLD VOLTAGE TOO HIGH - The specified threshold voltage is greater than 4.97 volts.

ATTEMPT TO OUTPUT TO INPUT PIN - An OUTPUT, CLOCKH or CLOCKL command was attempted to a pin not specified as an output by the last SETUP.

NO SUPPLY/GROUND PRESENT - No supply or ground was found by the SETUP command.

DIRECTORY FULL - The disk directory was full during a SAVE command.

DISK FULL - The disk was full during a SAVE command.

NO SUCH FILE - An attempt was made to LOAD a program that did not exist.

DRIVE NOT READY - The disk drive is not ready for operation, usually due to a missing disk. .

UNFORMATTED DISK - The disk media is unformatted or badly damaged.

SECTOR NOT FOUND ON DISK - The sector contained in the directory entry for the file cannot be located on the disk, usually due to a defective disk.

WRITE PROTECTED DISK - The write protect tab on the disk is in the protect position.

CRC ERROR ON DISK - The data on the disk cannot be read without errors, usually due to a defective disk.

CURSOR POSITION OFF SCREEN - The calculated cursor co-ordinates will result in the cursor position being off the screen

CANNOT DEFINE SUPPLY PINS AFTER SETUP - The SUPPLY command must come before the first SETUP command in the program.

DEFINED SUPPLY PIN IS TOO LARGE - The previous SUPPLY command defined a pin that was larger than the number of pins defined in the SETUP command.