I/OBASIC LANGUAGE

REFERENCE MANUAL

28 January 1987

Ninth Printing (rev) April 1986

The material in this manual is for informational purposes only and is subject to change without notice.

ADAC Corporation assumes no responsibility for any errors which may appear in this document.

Printed in U.S.A.

I/OBASIC Software rev level supported: V00.0C

CONTENTS

CHAPTER 1

INTRODUCTION

This manual provides a summary of commands and information necessary to use the I/OBASIC interpreter that runs on the ADAC BASYS systems. There are six BASYS systems available, DISKBASYS, PROMBASYS, DX11, PX11, MICROBASYS and PICOBASYS. If you are a first-time user, refer to the BASYS User's Guide for instructions on using the BASYS systems.

It is not necessary to read this manual from beginning to end. The information given here can be used for reference when you are interacting with the computer and when you are writing application programs.

1.1 INTENDED AUDIENCE

This manual is intended for experienced users and for users who have read the BASYS User's Guide. Some knowledge of standard BASIC is assumed, although a full description of I/OBASIC language elements is given in this manual.

1.2 RELATED DOCUMENTS

The BASYS System manuals provide sufficient information to enable first-time users to operate a BASYS System. The following additional manuals are suggested if more information is desired for operation of a BASYS System.

Digital Equipment Corporation RT-11 Manuals

Digital Equipment Corporation RT-11 BASIC Manual

1.3 STRUCTURE OF THIS MANUAL

This Reference Manual is divided into chapters which describe I/OBASIC language elements. These chapters are summarized below:

Chapter 2 - Keystroke Commands

Chapter 2 defines in alphabetical order the special control character commands used to interact with the BASYS System at the level of terminal input or output. These commands are used to correct typing mistakes before they are entered and to control output to the terminal.

Chapter 3 - RT-11 Commands

Chapter 3 defines in alphabetical order the commands used to interact with the operating system software, called RT-11. These commands are used to perform file maintenance operations such as deleting, copying or renaming files. RT-11 commands are applicable only to DISKBASYS and PROMBASYS systems.

Chapter 4 - I/OBASIC Enhancements

Chapter 4 defines the I/OBASIC programming features that differ from standard BASIC. These features are used to make your I/OBASIC programs more readable and easier to write and debug.

Chapter 5 - I/OBASIC Language Elements

Chapter 5 defines in alphabetical order all I/OBASIC programming language elements. These include standard BASIC statements, commands and functions, as well as statements in I/OBASIC that allow you to perform input and output (I/O) on analog and digital devices attached to your system. Example programs are also given for most of the language elements.

Appendix A - I/OBASIC Error Messages

Appendix A lists the error messages returned during I/OBASIC program execution because of errors in commands or statements. Both fatal errors and warnings are listed.

Appendix B - I/OBASIC Error Codes

Appendix B lists the numeric error codes that correspond to possible I/OBASIC program errors.

Appendix C - ASCII Equivalents

Appendix C lists the decimal and octal code equivalents of ASCII characters.

Appendix D - Device Names and File Extensions

Appendix D lists possible device names and file extensions used in RT-11 and I/OBASIC file specifications.

Appendix E - I/OBASIC Keywords

Appendix E contains all of the I/OBASIC keywords that cannot be used for variable names in an I/OBASIC program.

CHAPTER 2

KEYSTROKE COMMANDS

The commands described in this chapter are used at the terminal to interact with the BASYS System. These commands allow you to control input and output at the terminal by stopping and starting terminal scrolling and correcting typing mistakes before they are entered.

These keystroke commands can be used at the console serial channel when interacting with the BASYS System. Serial channels other than the console serial channel also support these keystroke commands when using the INPUT @, LINPUT @, and PRINT @ statements.

To specify a CTRL/x keystroke command, type the letter indicated while holding down the CTRL key.

CTRL/C

Stops a running I/OBASIC program and returns the user to I/OBASIC command level. Two CTRL/Cs are needed if an I/OBASIC program is executing, but only one CTRL/C is needed if an I/OBASIC program is waiting for terminal input at an INPUT statement. This keystroke command can only be issued at the console serial channel. It has no effect if issued at the other serial channels.

When CTRL/C is typed, it echoes as ^C followed by a carriage return and line feed.

CTRL/O

Inhibits the remainder of the output from printing on the terminal. This command might be used to end a lengthy LIST command.

When CTRL/O is typed, it echoes as ^O followed by a carriage return and line feed.

CTRL/Q

Causes terminal output to resume if it has been halted by a CTRL/S.

CTRL/S

Causes a pause or halt of the output to the user's terminal; output resumes with CTRL/Q.

CTRL/U

Cancels every character on the current line. This command can be used to cancel an incorrect line before it is entered so you can retype it.

When CTRL/U is typed, it echoes as ^U followed by a carriage return and line feed.

DELETE key

Deletes the last character entered. This key will also erase the character on the terminal by issuing a backspace, space, backspace sequence.

CHAPTER 3

RT-11 COMMANDS

The operating system software for DX11 and PX11 is called RT-11. RT-11 consists of a group of programs that allow you to control and interact with the resources of the computer. The I/OBASIC interpreter, for example, is a program that is run under the control of RT-11. The DX11 and PX11 Systems have been designed to minimize your interaction with RT-11, since most program development operations can be performed at the I/OBASIC interpreter level.

This chapter lists only the RT-11 commands you will need to operate the DX11 or PX11 Systems. The RT-11 commands described here are used to perform file maintenance functions, as well as several miscellaneous operations.

The following RT-11 commands are listed in alphabetical order. Where appropriate the system's response to the command is given. For more information about these RT-11 commands, see Chapter 4 of the BASYS User's Guide.

If an RT-11 command is entered incorrectly, an error message will be returned. The error messages are self-explanatory.

All RT-11 commands are terminated by pressing the carriage return key on the terminal keyboard. This key is symbolized by <ret> in the command formats given below.

ASSIGN <ret>

Assigns a logical device name to a physical storage device. For example, you can assign the logical device name DK: to a drive to specify that drive as the default drive for RT-11 commands. The system prompts for logical device name and physical device name.

COPY <ret>

Copies the specified file from one storage volume to another. The system will prompt for the input and output files.

COPY/WAIT <ret>

Copies the specified file from one storage volume to another when it is desired to temporarily remove the volume in a device unit. The system will prompt you to place the input and output volumes in the specified drives.

DATE <ret>

Prints the current system date if it has been set.

DATE dd-mmm-yy <ret>

Sets the current system date specified in the form day-month-year. For example, 02-FEB-83 specifies February 2nd, 1983.

DELETE [filespec1 [,filespec2 ...]] <ret>

Deletes specified file(s) from storage volume directory and makes the space they occupied available for reuse. Multiple files can be specified separated by commas. The system will confirm each file individually before deleting it. The system will prompt for filenames if none are specified.

DIRECTORY [ddn:] <ret>

Lists the directory of the volume in device ddn: on the terminal. The default device is DK: if no device is specified.

DIRECTORY/FULL [ddn:] <ret>

Lists the full directory of the volume in device ddn: on the terminal. The full directory lists both file names and free space arrangement. The default device is DK: if no other device is specified.

DIRECTORY/BRIEF [ddn:] <ret>

Lists the brief directory of the volume in device ddn: on the terminal. The brief directory gives only file names and file extensions in a 5-column format. The default device is DK: if no other device is specified.

DIRECTORY/PRINTER [ddn:] <ret>

Lists the directory of the volume in device ddn: on the line printer if a line printer is available. Make sure the line printer is turned on before issuing this command.

INITIALIZE [ddn:] <ret>

Clears the directory of the volume in device ddn:. The system will prompt for the device if it is not given. New volumes must be initialized prior to use.

RENAME [old file name] [,new file name] <ret>

Gives a new name to file. The system will prompt for a file name and the new name if they are not provided in the command line.

SQUEEZE [ddn:] <ret>

Moves the files on a device so that the free space is compressed into a contiguous block. This permits a larger usable free space to be on the device, and is necessary if the disk free space becomes 'chopped' by the creation or deletion of a large number of files. The system prompts for a device name if it is not provided in the command line.

TIME <ret>

Prints the current system time if it has been set.

TIME hh:mm:ss <ret>

Sets the current system time to the time specified in the command line. The time is supplied in 24-hour notation in the form hour:minute:second. For example, 14:10:00 specifies the time 2:10 p.m.

TYPE [filename] <ret>

Lists the contents of a file on the console terminal. This command may be used to list an I/OBASIC program, a data file, a command file, or any other file that is in ASCII format. It may not be used to list the contents of a binary file.

CHAPTER 4

I/OBASIC ENHANCEMENTS

This chapter describes briefly the enhancements to standard BASIC which are provided in I/OBASIC. For more information about these features see Chapter 5 of the BASYS User's Guide.

4.1 LONG VARIABLE NAME SUPPORT

I/OBASIC variable names can be up to 32 characters long and can contain letters, numbers and the underscore character. Variable names must begin with a letter. The following are examples of valid I/OBASIC variable names:

gas_flow, temp12, delay_4time

4.2 UPPER/LOWER CASE SUPPORT

I/OBASIC variable names will always appear in lower case when the program is listed, regardless of how they are input. For example, if you type the variable "GAS_FLOW", it will appear as "gas_flow" when the program is listed.

In addition, all I/OBASIC statement and function names, such as PRINT, GOTO, etc., will always appear in upper case when the program is listed, regardless of how they are typed in. For a complete listing of the keywords that always appear in upper case, see Appendix E.

4.3 PROGRAM FORMATTING SUPPORT

To improve readability of your I/OBASIC programs, you can insert any number of spaces and tabs between the program line number and the first word of the program line. This allows you to use an indented structure when you program to facilitate debugging and readability.

4.4 STATEMENT SHORT FORMS

Several I/OBASIC statements that perform analog and digital I/O have an abbreviated name that can also be used. The following are the statements that have both a long and a short form:

LONG FORM	SHORT FORM
ANALOG_IN	AIN
ANALOG_LOW_IN	AINL
ANALOG_OUT	AOT
BIT_CLEAR	BIC
BIT_SET	BIS
BIT_TEST	BIT
DIGITAL_IN	DIN
DIGITAL_OUT	DOT
TEMPERATURE_IN	TMPIN

4.5 INDIRECT COMMAND FILE SUPPORT

DX11 and PX11 Systems allow the I/OBASIC interpreter to read input lines from an indirect command file. This feature can be used for applications that execute I/OBASIC programs automatically on power up. See the BASYS User's Guide for more information on this feature.

CHAPTER 5

I/OBASIC PROGRAMMING ELEMENTS

This chapter describes I/OBASIC syntax and language elements. The language elements include statements, commands, and functions, which can be used to create I/OBASIC programs. These language elements include those found in standard BASIC, as well as additional statements that can be used for real-time data acquisition and control. This chapter is for reference purposes and is not designed as a tutorial.

5.1 I/OBASIC LINE FORMAT

An I/OBASIC program line consists of a line number, one or more statements, and a line terminator. An I/OBASIC program line cannot exceed 80 characters in length.

Every program line begins with a line number which must be an integer between 1 and 32767. Each line number must be unique. Any number of spaces and/or tabs may be inserted between the line number and the first character of the line to create an indented format. The program line should be terminated with a carriage return character.

An I/OBASIC program line can contain a single statement or several statements separated by backslash characters (\). For example:

```
320 LET alpha = beta + 5
330 PRINT alpha
or
320 LET alpha = beta + 5 \ PRINT alpha
```

When multiple statements occur on one program line, the line number refers to all the statements on the line.

5.2 I/OBASIC EXPRESSION OPERATORS

Expressions in I/OBASIC statements may contain arithmetic and logical operators. The following is a list of these operators in order of evaluation precedence (from first to last evaluated):

^	power
*, /	multiplication, division
+, -	addition, subtraction
NOT	logical negation
AND, OR	logical AND, inclusive OR
XOR	logical exclusive OR

Note that the string concatenation operator is the plus sign (+). Also, the logical operators convert their arguments to 16-bit integers before performing the operation.

The following are examples of the use of logical operators:

```
10 IF alpha% AND beta% = NOT gamma % THEN 200

10 DIGITAL_OUT(channel,value% XOR mask%)

10 PRINT NOT x_variable
```

5.3 BASYS FILE SPECIFICATIONS

All files residing on a DX11 or PX11 Systems are identified using the following format. The square brackets are used to indicate an optional part of the file specification:

```
[ddn:]filename[.ext]
```

ddn: is the device specification. The default specification is DK:. Device names are listed in Appendix D.

filename is the one to six character name of the file. This name can consist of any combination of letters A to Z and digits 0 to 9.

.ext is the three-letter file extension. Possible file extensions are given in Appendix D. A period should always precede the letters in the file extension.

5.4 STATEMENTS, COMMANDS, AND FUNCTIONS

I/OBASIC statements, commands, and functions are used to create, edit, and run I/OBASIC programs. These elements include those found in standard BASIC, as well as statements added to perform real-time analog and digital I/O. BASIC programs written to run on other systems can be used on your BASYS System with usually very little modifications.

I/OBASIC programming elements are listed here in alphabetical order. The syntax of the element is given followed by a description of the element and brief programming examples where appropriate.

The following documentation conventions are used in describing the I/OBASIC programming elements:

exp

Any valid I/OBASIC expression. A numeric expression is required unless otherwise specified.

filespec

A BASYS file specification as described in Section 5.2. Only DX11 and PX11 Systems support file specifications. MICROBASYS and PICOBASYS do not support file specifications.

integer

Any positive integer number constant or any positive numeric constant that could be an integer if it were followed by a percent sign.

linenumber

Any valid I/OBASIC program line number, such as 10, 65, 32767.

string

Any string expression such as "ABC", name$, or d$+SEG$(a$,3,4).

variable

A floating point, integer or string variable.

[ ]

The enclosed element is optional. For example,

```
[LET] variable = expression
```

In the statement above, the LET statement is optional and need not be specified.

...

Preceeding element can be repeated as indicated. For example,

```
DEL linenumber1 [,linenumber2 ...]
```

In the command line above, multiple line numbers may be specified on one command line.

All I/OBASIC command and statement lines are terminated by a carriage return unless specified otherwise in the text. In the examples that follow, the symbol <ret> is used to indicate a carriage return typed by the user.

```
+--------------------+
!      Function      !
!                    !
!       ABORT        !
+--------------------+
```

ABORT Function

SYNTAX: variable = ABORT(code)

DESCRIPTION:

The ABORT function causes termination of a program. The variable specified can be any valid I/OBASIC variable -- it is simply part of the syntax of the function and need not be defined.

The code specified can be either 1 or 0. If the value is 1, program execution will be stopped and the program will be cleared from immediate memory. If the value is 0, program execution will be stopped, but it will remain in immediate memory.

The ABORT function is not supported in MICROBASYS or PICOBASYS.

```
+--------------------+
!      Function      !
!                    !
!        ABS         !
+--------------------+
```

ABS Function

SYNTAX: ABS(numeric expression)

DESCRIPTION:

The ABS function returns the absolute value of the specified numeric expression. The absolute value of a positive number is equal to the number, but that of a negative number is equal to -1 times the number. Whether the expression is in integer or real number format, the returned function value is in floating point format.

EXAMPLE:

```
>listnh <ret>
10 xray = 72
20 PRINT ABS(xray)
30 PRINT ABS(-6.536)
40 PRINT ABS(3.50000E+21)

>runnh <ret>

 72
 6.536
 3.50000E+21

>
```

Statement
ANALOG_IN

ANALOG_IN Statement

SYNTAX: ANALOG_IN(chan, val [,flag])

or

AIN(chan, val [,flag])

DESCRIPTION:

The ANALOG_IN statement reads one or more channels on the high-level analog input board.

Argument chan is a variable or constant of any type that contains the analog channel to be read, or the last channel to scan if channel scanning is set. The value of this argument must be supplied by the program. Legal range is 0 to 63 for the first high-level board and the second (if any) is accessed by chan values 128-191. Chan values 0-63 correspond to channels 0-63 on the first board, and chan values 128-191 correspond to channels 0-63 on the second board. The boards operate independently with respect to channel scanning and DMA (if present), so it is possible to to have two ANALOG_IN statements with flags specified operating simultaneously.

Argument val is a variable of any type that will be returned with the value of the analog channel. If argument val is of type real the value is returned in units of either volts or percent full-scale. If argument val is of type integer or string, the value returned is the unconverted binary value of the A/D converter. If argument val is an array or array element then the entire array will be filled with analog channel values. String arrays are not allowed.

Argument flag is an optional integer or real variable. When specified, causes the statement to operate asynchronously in either interrupt mode or DMA mode (only if an ADAC 1622DMA controller is present). It will be set to a zero prior to the next I/OBASIC program statement, and will be set to a one when all of the analog values have been read. If argument flag is not specified then the statement will operate synchronously, so that all processing is completed before the next I/OBASIC statement. DMA is not supported in PICOBASYS hence argument flag is not used.

NOTE

The ANALOG_IN statement also supports direct transfer of analog input channel data to a virtual array. The virtual array must be located on an XM memory disk, and it must be of type integer. For complete information on this feature, see the chapter Data Files and Virtual Arrays in the BASYS User's Guide.

Statement
ANALOG_LOW_IN

ANALOG_LOW_IN Statement

SYNTAX: ANALOG_LOW_IN(chan, val [,flag])

or

AINL(chan, val [,flag])

DESCRIPTION:

The ANALOG_LOW_IN statement reads one or more channels on the low-level analog input board.

Argument chan is a variable or constant of any type that contains the analog channel to be read, or the last channel to scan if channel scanning is set. The value of this argument must be supplied by the program. Legal range is 0 to 1023 in DX11 and PX11.

Argument val is a variable of any type that will be returned with the value of the analog channel. If argument val is of type real the value is returned in units of either volts or percent full-scale. If argument val is of type integer or string, the value returned is the unconverted binary value of the A/D converter. If argument val is an array then the entire array will be filled with analog channel values. String arrays are not allowed.

Argument flag is an optional integer or real variable. When specified, causes the statement to operate asynchronously in interrupt mode, or DMA mode if an ADAC 1622DMA controller is present. It will be set to a zero prior to the next I/OBASIC program statement, and will be set to a one when all of the analog values have been read. If argument flag is not specified then the statement will operate synchronously, so that all processing is completed before the next I/OBASIC statement.

NOTE

The ANALOG_LOW_IN statement also supports direct transfer of analog input channel data to a virtual array. The virtual array must be located on an XM memory disk, and it must be of type integer. For complete information on this feature, see the chapter Data Files and Virtual Arrays in the BASYS User's Guide.

Statement
ANALOG_OUT

ANALOG_OUT Statement

SYNTAX: ANALOG_OUT(chan, val [,flag])

or

AOT(chan, val [,flag])

DESCRIPTION:

The ANALOG_OUT statement writes to a specified analog output channel.

Argument chan is a variable or constant of any type that contains the analog output channel to be written. The value of this argument must be supplied by the program. Legal range is 0 to 127 for DX11 and PX11 systems.

Argument val is a variable or constant of any type that contains the value(s) to be written to the analog output channel. If argument val is of type real the value is in units of either volts or percent full-scale. If argument val is of type integer or string, the value is the unconverted binary value for the D/A converter. If val is an array, then the entire array will be written to the analog channel. String arrays are not allowed. The value(s) for this argument must be supplied by the program.

Argument flag is an optional integer or real variable. When specified, causes the statement to operate asynchronously in DMA mode if an ADAC 1622DMA controller is present and the channel is part of an ADAC 1023EX board. If flag is specified and no DMA hardware is present, then an error message results. It will be set to a zero prior to the next I/OBASIC program statement, and will be set to a one when all of the analog values have been written. If argument flag is not specified then the statement will operate synchronously in program control mode, so that all processing is completed before the next I/OBASIC statement. If this argument is specified, then argument val must be of type integer, with values in the following range:

1. zero to 4,095 for full scale unipolar 12 bits;

2. -2,048 to +2,047 for full scale bipolar 12 bits.

NOTE

The ANALOG_OUT statement also supports direct transfer of data from a virtual array to an analog output channel. The virtual array must be located on an XM memory disk, and it must be of type integer. For complete information on this feature, see the chapter Data Files and Virtual Arrays in the BASYS User's Guide.

Command
APPEND

APPEND Command

SYNTAX: APPEND file specification

DESCRIPTION:

The APPEND command loads the specified program file into immediate memory and merges it with the program already in immediate memory. APPEND cannot be used on a compiled file. If common line numbers exist, the appended program lines will replace the original lines.

EXAMPLE:

```
        Program 1                       Output 1

        10 REM This is PROG1            >runnh <ret>
        30 time = 50                     50
        40 PRINT time
        50 END

        Program 2                       Output 2

        10 REM This is PROG2            >runnh <ret>
        20 volts = 10                    100
        50 PRINT volts * 10
        60 END
```

Program 1 is in immediate memory. If you enter the command: APPEND PROG2, the following program is the result:

```
10 REM This is PROG2     <Line 10 replaced by Program 2 line 10
20 volts = 10            <Line 20 from Program 2
30 time = 50             <Line 30 from Program 1
40 PRINT time            <Line 40 from Program 1
50 PRINT volts * 10      <Line 50 replaced by Program 2 line 50
60 END                   <Line 60 from Program 2

        >runnh <ret>

         50
         100

        >
```

Function
ASC

ASC Function

SYNTAX: ASC(character string)

DESCRIPTION:

The ASC function is used to convert a one-character string to its ASCII value. The integer value of the decimal ASCII code is returned for the character specified. If the string is null or contains more than 1 character, I/OBASIC returns the error message: "?Argument error". Appendix C lists the decimal and octal equivalents for ASCII characters.

EXAMPLE:

```
>listnh <ret>
10 PRINT "Enter any character followed by a return"
20 INPUT any_character$
30 number = ASC(any_character$)
40 PRINT "The ASCII value of that character is:";number
50 END

>runnh <ret>

Enter any character followed by a return?
B <ret>

The ASCII value of that character is: 66

>
```

Function
ATN

ATN Function

SYNTAX: ATN(numeric expression)

DESCRIPTION:

The ATN function returns the arctangent of the expression specified as an angle in radians in the range +PI/2 to -PI/2.

EXAMPLE:

```
>listnh <ret>
10 PRINT "Enter a number";
20 INPUT num
30 angle = ATN(num) / PI * 180
40 PRINT "The angle is ";angle
50 END

>runnh <ret>

Enter a number? 1 <ret>
The angle is  45

>runnh <ret>

Enter a number? 30 <ret>
The angle is  88.0909

>
```

```
+--------------------+
!      Function      !
!                    !
!        BIN         !
+--------------------+
```

BIN Function

SYNTAX: BIN(string of binary digits)

DESCRIPTION:

The BIN function returns the decimal value of the binary string specified. Spaces which occur in the expression are ignored and the value is returned as an integer.

The binary number is treated as a signed 2's complement integer and its absolute value may not be larger than 2^15-1

EXAMPLE:

```
>listnh <ret>
10 PRINT "Enter a binary number";
20 INPUT bin$
30 num = BIN(bin$)
40 PRINT "The decimal equivalent";num
50 END

>runnh <ret>

Enter a binary number? 10 <ret>
The decimal equivalent 2

>runnh <ret>

Enter a binary number? 11101 <ret>
The decimal equivalent 29

>
```

```
+--------------------+
!      Statement     !
!                    !
!     BIT_CLEAR      !
+--------------------+
```

BIT_CLEAR Statement

SYNTAX: BIT_CLEAR(chan, bit)

or

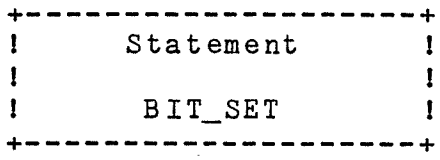
BIC(chan, bit)

DESCRIPTION:

The BIT_CLEAR statement clears a single bit in a digital output channel.

Argument chan is a variable or constant of any type that contains the digital channel for the bit to be cleared. The value of this argument must be supplied by the program. Legal range is 0 to 127 for DX11 and PX11 systems.

Argument bit is a variable or constant of any type that contains the bit number to be cleared in the channel. The value of this argument must be supplied by the program. Legal range is 0 to 15.



BIT_SET Statement

SYNTAX: BIT_SET(chan, bit)

or

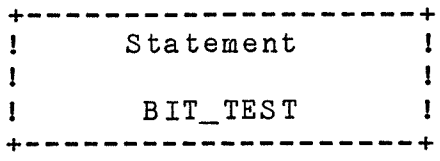
BIS(chan, bit)

DESCRIPTION:

The BIT_SET statement sets a single bit in a digital output channel.

Argument chan is a variable or constant of any type that contains the digital channel for the bit to be set. The value of this argument must be supplied by the program. Legal range is 0 to 127 for DX11 and PX11 systems.

Argument bit is a variable or constant of any type that contains the bit number to be set in the channel. The value of this argument must be supplied by the program. Legal range is 0 to 15.



BIT_TEST Statement

SYNTAX: BIT_TEST(chan, bit, val)

or

BIT(chan, bit, val)

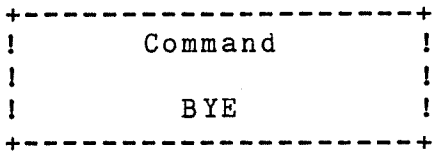
DESCRIPTION:

The BIT_TEST statement will test a single bit in a digital input or output channel.

Argument chan is a variable or constant of any type that contains the digital channel for the bit to be tested. The value of this argument must be supplied by the program. Legal range is 0 to 127 for DX11 and PX11 systems.

Argument bit is a variable or constant of any type that contains the bit number to be tested in the channel. The value of this argument must be supplied by the program. Legal range is 0 to 15.

Argument val is a variable of any type that will be set to the state of the bit. If the bit is on, argument val will equal a one, and if the bit is off, argument val will equal a zero.



BYE Command

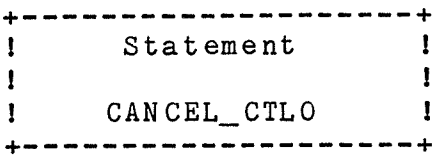
SYNTAX: BYE

DESCRIPTION:

The BYE command exits from the I/OBASIC interpreter and returns the user to control of the RT-11 operating system.

EXAMPLE:

```
>bye <ret>

.                    <RT-11 prompt is printed
```



CANCEL_CTLO Statement

SYNTAX: CANCEL_CTLO

DESCRIPTION:

The CANCEL_CTLO statement will cancel the effect of a CTRL/O command. It causes output to resume printing at the console terminal after a CTRL/O has discarded the output.

CTRL/O is a binary command key, so that typing it once will cause output sent to the console terminal to be discarded, and typing it again will cause output to resume printing.

The CANCEL_CTLO statement can be useful if executed prior to printing a summary line within a program, so that the user can skip lengthy intermediate output with a CTRL/O if desired, and still get the summary line at the end.

CHAIN Statement

SYNTAX: CHAIN "string" [LINE line number]

DESCRIPTION:

The CHAIN statement allows you to separate larger programs into smaller subprograms so that at any one time only a portion of a large program is in immediate memory. When a program is segmented, it has lower memory requirements; this means that programs that exceed memory size can be broken up and run in segments.

Segmenting programs also simplifies debugging since the segments can be run independently for testing.

When the CHAIN statement is executed, the current program in execution is terminated and erased from immediate memory; the program specified by the string in the CHAIN statement is loaded into immediate memory and execution begins at the line number specified. If no line number is specified, execution begins at the lowest line number.

The program segment includes all program lines, user defined functions, arrays, and variables except those that are listed in COMMON statements. The COMMON statement, used in conjunction with the CHAIN statement, preserves variables specified in one segment through another.

If the file extension is not specified, the CHAIN command looks for the compiled version (.BAC extension) first. If it cannot find it, it then looks for the non-compiled version (.BAS extension).

CHAR_IN Statement

SYNTAX: CHAR_IN(chan, var)

DESCRIPTION:

The CHAR_IN statement will read one or more characters from a serial channel. This statement can be used to read characters already contained in the input buffer for a serial channel. The CHAR_IN statement is different from the INPUT @ and LINPUT @ statements, since it does not wait for input, but rather reads one or more characters already typed at the serial channel specified. It is therefore useful for polling serial channels for input, without causing the program to 'hang' at one spot.

When the CHAR_IN statement is executed, all characters subsequently typed at the serial channel will not be echoed until an INPUT @ or LINPUT @ statement is executed for that serial channel (or INPUT and LINPUT statements for the console serial channel).

Argument chan is a variable or constant of any type that contains the serial channel number. The value of this argument must be supplied by the program. Legal range is 0 to 7 for DX11 and PX11. Note that serial channel number zero is the console terminal.

The CHAR_IN statement acts differently depending whether the type of argument var is numeric (integer or real) or string.

If argument var is of type string, all characters currently residing in the input buffer for the serial channel are placed in the string. More than one character can therefore be read using a string variable. If no characters are present in the input buffer when the CHAR_IN statement executes a null string will be returned.

If argument var is of type integer or real, only one character will be read from the input buffer. Its ASCII code equivalent will be returned in the numeric variable. If no characters are present in the input buffer when the CHAR_IN statement executes, a -1 will be returned.

EXAMPLE:

```
>listnh <ret>
10 REM poll serial channels and send input to console
20 FOR term_no = 1 to 3
30 CHAR_IN(term_no,a$)
40 IF a$ = "" GO TO 60
50 PRINT "Channel ";term_no;" ";a$
60 NEXT term_no
70 END

>
```

Function
CHR$

CHR$ Function

SYNTAX: CHR$(ASCII value expression)

DESCRIPTION:

The CHR$ function converts the ASCII value expression specified and returns a one-character string equivalent. The expression must be between 0 and 255. I/OBASIC treats arguments greater than 255 modulo 256, therefore, 256 is treated as 0, 257 as 1, etc.

EXAMPLE:

```
>listnh <ret>
10 PRINT "Enter an ASCII value for 1 character";
20 INPUT number
30 IF number > 31 THEN 60
40 PRINT "That would be a control.  Try again."
50 GO TO 10
60 IF number < 127 THEN 90
70 PRINT "That is outside the upper limit.  Try again."
80 GO TO 10
90 PRINT "The character ";CHR$(number);" is equal to";
100 PRINT number
110 END

>runnh <ret>

Enter an ASCII value for 1 character? 7 <ret>
That would be a control.  Try again.
```

```
Enter an ASCII value for 1 character? 200 <ret>
That is outside the upper limit.  Try again.

Enter an ASCII value for 1 character? 65 <ret>
The character A is equal to 65

>
```

Command
CLEAR

CLEAR Command

SYNTAX: CLEAR

DESCRIPTION:

The CLEAR command removes all common and noncommon variables from immediate memory, initializes all numeric variables to zero and all string variables to nulls. Arrays are also deleted. Program text and subroutine stacks remain unaffected. When CLEAR is executed, the amount of space formerly used by arrays is available for additional program text.

Function
CLK$

CLK$ Function

SYNTAX: CLK$

DESCRIPTION:

The CLK$ function returns the present time in hours, minutes and seconds in 24-hour notation.

The time can be set with the I/OBASIC SET_TIME statement.

EXAMPLE:

```
>listnh <ret>                                  12:00:14
10 FOR alpha  = 1 TO 6                         12:00:15
20      PRINT CLK$                             12:00:16
30      WAIT(1)                                12:00:17
40 NEXT alpha                                  12:00:18
50 END                                         12:00:19
>runnh <ret>                                   >
```

```
+--------------------+
!     Statement      !
!                    !
!     CLOCK_OUT      !
+--------------------+
```

CLOCK_OUT Statement

SYNTAX: CLOCK_OUT(rate, val [,flag])

DESCRIPTION:

The CLOCK_OUT statement operates the ADAC 1601GPT real-time clock on DX11 and PX11.

Argument rate is a variable or constant of any type that indicates the rate for the real-time clock. The value of this argument must be supplied by the program. Legal range is 0 to 7. The table below indicates what rates are selected by different values of argument rate:

RATE	MEANING
0	Stop
1	1 MHz
2	100KHz
3	10 KHz
4	1 KHz
5	100 Hz
6	EXT CLK (user supplied)
7	60 Hz (EVENT CLK), 10 Hz

Argument val is a variable or constant of any type but that is a whole number. Val is the number (range 1 to 32767) of units that the clock is to count before it resets. Divide the value for argument rate by the value for argument val to find the number of pulses delivered by the hardware in the given time period (specified by argument rate).
For example:

CLOCK_OUT(2,1000) causes:
100,000(units/second)/1000(units/pulse) = 100 pulses/second.

The argument flag is an optional real or integer variable. If the argument flag is absent then the clock operates in Mode 3 (see hardware reference manual) and pulses are generated after every "val" counts as described above until the clock is stopped with the statement CLOCK_OUT(0). In this mode the CLOCK_OUT() statement operates synchronously and the statements following CLOCK_OUT() will not execute until the CLOCK_OUT() related processing is completed.
For example:

```
30 DIM a%(1000)
40 CLOCK_OUT(2,10)
50 ANALOG_IN(6,a%(),done)
60 PRINT "MESSAGE"
```

"MESSAGE" will get printed when array a% is filled with 1001 integers from analog channel 6. This data is being converted at 100000/10 = 10000 samples per second thus 2% will fill in 0.10 second.

If argument flag is specified then the GPT will operate in Mode 2 (see hardware manual). A single pulse and an interrupt will be generated at the end of the time interval.
For example:

```
20 CLOCK_OUT(2,10000,ready)
30 PRINT "COUNTING"
40 ANALOG_IN(5,pressure)
```

The clock will commence counting 10,000 counts at 100,000 counts/sec., that will take 10,000/100,000 = 0.10 second. Asynchronously the word "COUNTING" will appear, program control will pause at line 40 awaiting the clock pulse to trigger the A to D at channel 5 which value will be stored in "pressure" and the value of ready will become = 1.

HINT:Read about DMA control and about the SET_AIN_TRIGGER statement.

Statement
CLOSE

CLOSE Statement

SYNTAX: CLOSE [[#]expression1, [[#]expression2 ...]]

DESCRIPTION:

The CLOSE statement closes the files associated with the channel specified in the expression. If no expression is specified, all open files are closed. Once a file is closed, it must be reopened to be accessed again.

Although all files are closed when an END statement, a CHAIN statement, or the highest numbered program line is executed, it is good practice to close all files that are opened by a program with the CLOSE statement.

Statement
COMMON

COMMON Statement

SYNTAX: COMMON variable1 [,variable2 ...]

DESCRIPTION:

The COMMON statement can be used to preserve variables when an I/OBASIC program chains to another I/OBASIC program with the CHAIN statement. The variables or arrays listed in a COMMON statement will retain their values after CHAIN is executed. The COMMON statement is an alternative to using data files for passing information between programs.

NOTE

The COMMON statement should appear in both programs, and it should have the same variable names, data types, and array dimensions listed in each.

In the syntax above, variable is any integer, real, or string variable or array. More than one COMMON statement can appear in an I/OBASIC program.

The variables that appear in a COMMON statement should be structured so that the same storage is allocated in the COMMON statements of both programs.

The program sections listed below provide a sample of how to set up COMMON statements for two programs. Assume that program 1 will chain to program 2. When program 2 starts execution, the variables declared in its COMMON statements will have the same values as those in the COMMON statements in program 1.

EXAMPLE:

Program 1

```
10 COMMON alpha, beta, gamma
20 COMMON int_array%(100)
30 COMMON real_array(256)
```

Program 2

```
100 COMMON alpha, beta, gamma
200 COMMON int_array%(100), real_array(256)
```

Command
COMPILE

SYNTAX: COMPILE file specification

DESCRIPTION:

The COMPILE command saves a compiled version of the program specified. A compiled version occupies less file space on a disk than an uncompiled version, but it is stored in a binary format that does not permit direct editing or viewing of the file. A compiled program will load into memory faster than an uncompiled program. Compiled programs have a ".BAC" extension.

Statement
CONVERT_OCTAL

CONVERT_OCTAL Statement

SYNTAX: CONVERT_OCTAL(arg1, arg2)

DESCRIPTION:

The CONVERT_OCTAL statement converts values from decimal to octal and vice versa. The decimal argument is specified as an integer or real and the octal argument is specified as a string. The input argument arg1 is converted to output argument arg2.

Argument arg1 is a constant or variable of any type. The value of this argument must be supplied by the program. If the type is

integer or real then it is treated as a decimal number. If the type is string then the string must contain a valid octal number representation. Legal range is 0 to 177777 (octal).

Argument arg2 is a variable of any type that will be returned with the same value as argument arg1. If arg1 is an octal string, then arg2 should be of type integer or real, and an octal to decimal conversion will occur. If arg1 is of type integer or real, then arg2 should be of type string, and a decimal to octal conversion will take place.

Function
COS

COS Function

SYNTAX: COS(numeric expression)

DESCRIPTION:

The COS function returns the cosine of the angle specified. The angle is specified in radians.

EXAMPLE:

```
>listnh <ret>
10   REM Convert angle to radians and print cosine
20 PRINT "Enter an angle in degrees";
30 INPUT angle
40   REM Convert the angle to radians
50 LET num = angle * 2 * PI / 360
60 PRINT "The cosine of angle A is ";COS(num)
70 END

>runnh <ret>
Enter an angle in degrees? 30 <ret>
The cosine of angle A is  .866025

>runnh <ret>

Enter an angle in degrees? 90 <ret>
The cosine of angle A is  0
>
```

Function
CTLC

CTLC Function

SYNTAX: CTLC

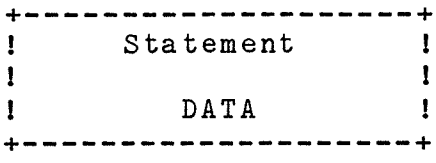
DESCRIPTION:

The CTLC function is used to indicate whether a CTRL/C (control-C) has been typed at the console terminal (serial channel number zero). The function returns a 1 if a CTRL/C has been typed, and a zero if it has not been typed. This function is useful when used with the DISABLE_CTLC statement to monitor the user's attempt to terminate the program.

See descriptions of the DISABLE_CTLC and ENABLE_CTLC statements for more information on CTRL/C processing.

EXAMPLE:

```
>listnh <ret>
10 DISABLE_CTLC
20 IF CTLC = 1 THEN 100
30 PRINT "Still Executing!"
40 WAIT (1)
50 GO TO 20
100 PRINT "Program terminating."
110 END
>
```



DATA Statement

SYNTAX: DATA value1 [,value2 ...]

DESCRIPTION:

The DATA statement supplies the values to be used in a READ statement. Each time a READ statement is executed the next sequential value in the DATA statement is assigned to the variable in the READ statement argument list. Since the values are taken sequentially, the order of values is important.

Literal strings or numeric variables may be used, but values specified in the DATA statement must correspond to variables in the READ statement. Commas separate all constants specified in the DATA statement.

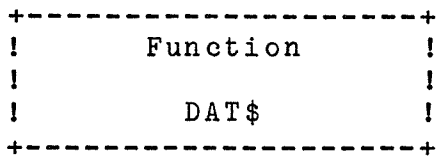
A program can have more than one DATA statement. DATA statements which have no corresponding READ statements are ignored.

All characters between the DATA statement and the next line number are treated as data. A DATA statement should, therefore, be the only statement on a line.

I/OBASIC ignores excess data in DATA statements, but variables in a READ statement with no corresponding value in a DATA statement result in an I/OBASIC error message, "?Out of data".

EXAMPLE:

```
>listnh <ret>
10 DATA "Tom Snyder",50,"Rona Barret",55
20 FOR count = 1 TO 2
30    READ person$,age
40    message$ = "years old."
50    PRINT count") ";person$;" is ";age;" "message$
60 NEXT count
70 END

>runnh <ret>

 1) Tom Snyder is  50  years old.
 2) Rona Barret is  55  years old.

>
```



DAT$ Function

SYNTAX: DAT$

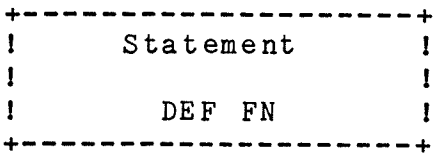
DESCRIPTION:

The DAT$ function returns the Gregorian date as a string in the form: DD-MON-YR.

The date can be set with the I/OBASIC SET_DATE statement.

EXAMPLE:

```
>PRINT "Today's date is "; DAT$ <ret>
Today's date is 21-JAN-83

>
```



DEF FN Statement

SYNTAX: DEF FNletter[$ or %] (list)=expression

DESCRIPTION:

The DEF FN statement declares a user-defined function. In the syntax above, list is any argument list, consisting of variable names or constants separated by commas. Expression is any valid mathematical or string expression. Letter is any single letter, A to Z. If a dollar sign follows the letter, then the function will return a string, and if a percent sign follows the letter, then the function will return an integer. If neither a dollar sign or percent sign is used, then the function returns a real number.

You must declare each user-defined function once in a program with a DEF FN statement. You can define it anywhere in the program. Ensure that expression is the same data type (string, integer, or real) as indicated by the function name.

Once you have defined the function anywhere in the program, you can use the function by specifying its name and argument list. The value of the function will be computed by substituting the variables in the argument list into the expression given in the DEF FN statement.

The defining expression can contain any constants, variables, or functions, as well as another user-defined function.

For example, the line:

```
10 DEF FNA(arg) = arg*2
```

causes a later statement:

```
20 PRINT FNA(4)
```

to print the number 8.

```
+--------------------+
!      Command       !
!                    !
!        DEL         !
+--------------------+
```

DEL Command

SYNTAX: DEL line specification1 [,line specification2 ...]

DESCRIPTION:

The DEL command deletes lines from a program in immediate memory. Individual lines can be deleted by entering the line number and a carriage return; multiple lines and/or a range of lines can be also be deleted with a single DEL command. Line numbers do not need to be specified in numerical order, and the same DEL command may contain single line numbers and ranges.

EXAMPLE:

```
>del 20,30,40 <ret>      <Deletes lines 20, 30, and 40

>del 20-40 <ret>         <Deletes lines 20 through 40

>del 40,20-25,30 <ret>   <Deletes line 40, lines 20
                          through 25, and line 30.
```

```
+--------------------+
!     Statement      !
!                    !
!    DIGITAL_IN      !
+--------------------+
```

DIGITAL_IN Statement

SYNTAX: DIGITAL_IN(chan, val [,flag])

or

DIN(chan, val [,flag])

DESCRIPTION:

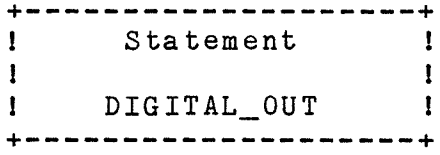
The DIGITAL_IN statement reads a digital input channel.

Argument chan is a variable or constant of any type that contains the digital input channel to be read. The value of this argument must be supplied by the program. Legal range is 0 to 127 for DX11 and PX11 systems.

Argument val is a variable or constant of any type that will be returned with the 16-bit value(s) of the digital input channel. If val is an array, then the entire array will be filled with values read from the input channel. String arrays are not allowed.

In the case where argument flag is specified in the statement, argument val is restricted to being of type integer.

Argument flag is an optional integer or real variable. When specified, causes the statement to operate asynchronously in interrupt mode, provided the hardware for the digital channel supports interrupts. For each hardware generated interrupt, the channel will be read and its value placed in argument val. When argument val has been filled, the flag will be set from a zero to a one. If argument flag is not specified then the statement will operate synchronously in program control mode, so that all processing is completed before the next I/OBASIC statement.



DIGITAL_OUT Statement

SYNTAX: DIGITAL_OUT(chan, val [,flag])

or

DOT(chan, val [,flag])

DESCRIPTION:

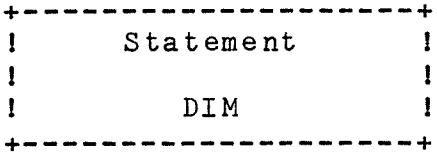
The DIGITAL_OUT statement writes to a digital output channel.

Argument chan is a variable or constant of any type that contains the digital output channel to be written. The value of this argument must be supplied by the program. Legal range is 0 to 127 for DX11 and PX11 systems.

Argument val is a variable or constant of any type that contains the value(s) to be written to the digital output channel. If argument val is an array, then the entire array will be written to the output channel. String arrays are not allowed. The value of this argument must be supplied by the program. Legal range is 0 to 177777 (octal).

In the case where argument flag is specified in the statement, argument val is restricted to being of type integer.

Argument flag is an optional integer or real variable. When specified, causes the statement to operate asynchronously in interrupt mode, provided the hardware for the digital channel supports interrupts. For each hardware generated interrupt, the channel will be written using the (next) value in argument val. When argument val has been entirely written, the flag will be set to a one. If argument flag is not specified then the statement will operate synchronously in program control mode, so that all processing is completed before the next I/OBASIC statement.



DIM Statement

SYNTAX: DIM variable1(integer1 [,integer2]) ,...

DESCRIPTION:

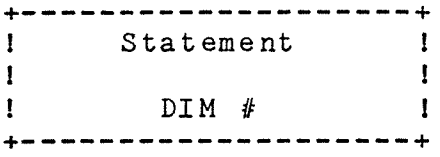
The DIM statement sets up the dimensions of an array in an I/OBASIC program. The array can be of type real, integer, or string, depending on the type of variable1 specified in the syntax above.

More than one array can be dimensioned in a DIM statement. In the syntax above, if optional integer2 is not specified, then the array is one-dimensional and is integer1+1 elements long. If integer2 is specified, then a two-dimensional array is specified, of size (integer1+1) x (integer2+1).

A DIM statement is not executed and can be placed anywhere in an I/OBASIC program. Multiple DIM statements can also be used within the same program. Note that all arrays start at element zero, rather than element one.

The following are some sample program lines containing DIM statements.

```
10 DIM alpha(100), beta(200)
20 DIM names$(15), data%(500)
```



DIM # Statement

SYNTAX: DIM #integer1, array [integer2]

DESCRIPTION:

The DIM # statement declares a virtual array. A virtual array is an array whose data is actually stored in a disk file rather than in working memory. The advantage of a virtual array is that it can have a larger size than a memory array.

In the syntax above, integer1 is a constant that specifies the channel number of the file containing the virtual array. In the DIM # statement above, array is the name of a one or two-dimensional array. It has the same format as in the standard DIM statement. Integer2 is an optional constant that specifies the maximum length for elements in a virtual string array.

A DIM # statement should always be followed by an OPEN statement to open the virtual array file with the same channel number.

See the chapter Using Data Files in the BASYS User's Guide for a complete description of the DIM # statement.

Statement
DISABLE_CTLC

DISABLE_CTLC Statement

SYNTAX: DISABLE_CTLC

DESCRIPTION:

The DISABLE_CTLC statement will prevent a CTRL/C typed at the console terminal from stopping an I/OBASIC program. Normally, a running program can be stopped by typing two CTRL/Cs, and a program waiting for terminal input can be stopped by typing one CTRL/C. After a DISABLE_CTLC statement is executed, a CTRL/C typed at the console terminal will have no effect on an executing I/OBASIC program.

This statement is useful for preventing critical portions of a program from being interrupted by the user.

See descriptions of the CTLC function and ENABLE_CTLC statement for more information on CTRL/C processing.

The ENABLE_CTLC statement reverses the effect of the DISABLE_CTLC statement, so that a CTRL/C can terminate a program.

Statement
ENABLE_CTLC

ENABLE_CTLC Statement

SYNTAX: ENABLE_CTLC

DESCRIPTION:

The ENABLE_CTLC statement reverses the effect of the DISABLE_CTLC statement, so that CTRL/C typed at the console terminal can terminate an executing I/OBASIC program. When a program is run, the ENABLE_CTLC statement is in effect, so that a CTRL/C may stop an executing program.

See descriptions of the CTLC function and DISABLE_CTLC statement for more information on CTRL/C processing.

Statement
END

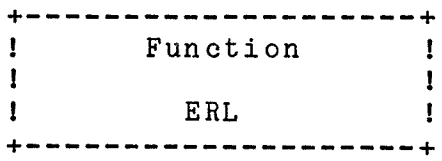
END Statement

SYNTAX: END

DESCRIPTION:

The END statement is optional. It can be used as the final line in a program to terminate execution. If END is used, it must occur on the highest line number of the program, since execution will cease when the system reads an END. No subsequent line numbers will be processed.

Only one END statement may be included in a program. If no END statement is included, the program will terminate upon execution of the highest line number. When the END statement is executed, the system closes all open files.



ERL Function

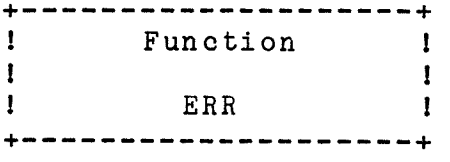
SYNTAX: ERL

DESCRIPTION:

The ERL function returns the value of the line number at which a program error occured. Error processing must have been previously activated by executing an ON ERROR GOTO statement prior to using the ERL function.

The ERL function is used within an error recovery routine in an I/OBASIC program to determine where in a program an error occured. This is useful when the same error condition can occur at different lines and requires different recovery procedures.

See descriptions of the ERR function and the ON ERROR GOTO and RESUME statements for additional information about error processing.



ERR Function

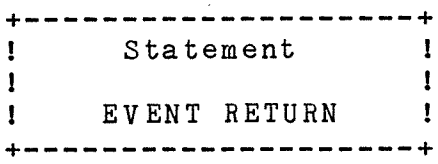
SYNTAX: ERR

DESCRIPTION:

The ERR function returns the value of the error code for the most recent program error. A list of possible error codes is given in Appendix B.

The ERR function is used within an error recovery routine in an I/OBASIC program to determine what type of program error occured. The error recovery routine can then make use of this information to select the proper recovery method.

See descriptions of the ERL function and the ON ERROR GOTO and RESUME statements for additional information about error processing.



EVENT RETURN Statement

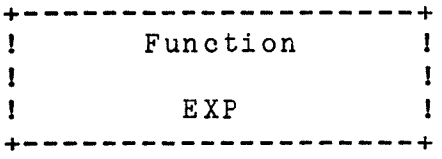
SYNTAX: EVENT RETURN

DESCRIPTION:

The EVENT RETURN statement causes an I/OBASIC program to resume execution at the point it was interrupted at when an event occured. Events are indicated by the setting of any flag argument specified in a real-time control statement (such as ANALOG_IN). The EVENT RETURN statement is used in conjunction with the ON EVENT GOSUB statement and the real-time control statements that support flags.

An EVENT RETURN statement must be the last statement executed within an event processing subroutine. It tells the system that further events (setting of flag arguments) can cause program execution to branch to the line number specified in an ON EVENT GOSUB statement. Events that occur while an event processing subroutine is executing are queued, so that after an EVENT RETURN is executed, then next (if any) event is processed.

See the description of the ON EVENT GOSUB statement for additional information about event processing.



EXP Function

SYNTAX: EXP(numeric expression)

DESCRIPTION:

The EXP function returns the value of e (approximately 2.7182818) raised to the power of the specified expression. For example, EXP(x) finds the number Y whose logarithm X is known.

The EXP function is the inverse of the LOG function.

EXAMPLE:

```
>listnh <ret>
10 PRINT "Input an exponent of e";
20 INPUT number
30 PRINT "e raised to the power ";number;
40 PRINT " equals ";EXP(number)
50 END

>runnh <ret>

Input an exponent of e? 1 <ret>
e raised to the power  1  equals  2.71828

>runnh <ret>

Input an exponent of e? 3 <ret>
e raised to the power  3  equals  20.0855

>
```

Statement
FOR-TO-NEXT

FOR-TO-NEXT Statement

SYNTAX:

```
FOR variable = beginning TO end [STEP inc]
  .
  .
 Other Statements
  .
  .
NEXT variable
```

DESCRIPTION:

The FOR-TO-NEXT I/OBASIC statements are used together to create a counted loop that terminates when the end value is reached. For example, the statement, FOR X=1 TO 10, is used to specify 10 processing loops.

The beginning count may be either a constant, integer variable or numeric variable. The beginning count should be the low value of the counter loop.

The end count specifies the point at which the loop ends. It should be the high value of the counter loop.

The NEXT statement occurs at the end of the processing loop and directs the computer to begin processing the next value for the variable in the loop. The NEXT variable and the FOR variable should be identical.

Each time the loop cycles, the NEXT statement increments the counter variable by the value of inc, or by one if STEP inc was not specified, and tests whether the end count has been reached. If the end count has not been reached, control returns to the FOR statement. If the end count has been reached, execution proceeds with the first executable statement following the NEXT statement.

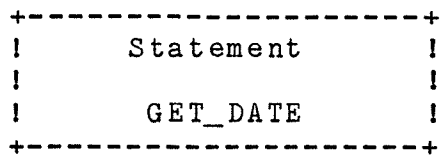
The STEP increment is optional and defaults to 1 if not specified. The STEP increment can be a positive or negative, integer variable or numeric variable. The beginning count will be the first value of the loop, followed by beginning count + step increment. In other words, FOR X=1 TO 10 STEP 3, will result in the loop being processed for the values 1, 4, 7, and 10.

The FOR-TO and NEXT statements must always be paired in a program, otherwise an error will result in execution. There is no limit to the number of lines of processing that may separate the two statements. FOR-TO-NEXT loops may be nested, that is, you may have a loop within a loop, but the counter variables must be unique so that the correct counter will increment for each loop.

EXAMPLE:

```
>listnh <ret>
10 REM Simple FOR-TO-NEXT Loop
20 FOR beta = 0 TO 10 STEP 2
30 PRINT beta * 5
40 NEXT beta
50 END

>runnh <ret>

 0
 10
 20
 30
 40
 50

>old prog4 <ret>

>listnh <ret>
10 begcount = 3
20 endcount = 1
30 FOR count = begcount TO endcount STEP  - 1
40 PRINT "This is line: ";count
50 NEXT count
60 PRINT "Loop is complete."
70 END
```

```
>runnh <ret>

This is line:  3
This is line:  2
This is line:  1
Loop is complete.

>
```



GET_DATE Statement

SYNTAX: GET_DATE(month, day, year)

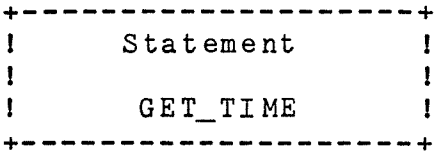
DESCRIPTION:

The GET_DATE statement will return the current system date.

Argument month is a variable of any type that will be returned with the value of the current month of the year.

Argument day is a variable of any type that will be returned with the value of the current day of the month.

Argument year is a variable of any type that will be returned with the value of the current year since 1900.



GET_TIME Statement

SYNTAX: GET_TIME(val)

DESCRIPTION:

The GET_TIME statement will return the current system time.

Argument val is a variable of any type that will be returned with the system time in seconds past midnight. If argument val is of type real, then fractional seconds can be returned. System time is kept in units of 1/60th of a second, and is accurate to 1/10th of a second when returned by this statement.

NOTE

If argument val is of type integer or string and the time is greater than 65,535 seconds past midnight a numeric overflow error will result. To avoid this, specify argument val as a real variable.

Statement
GOSUB

GOSUB Statement

SYNTAX: GOSUB line number

DESCRIPTION:

The GOSUB statement is used to transfer control to a subroutine within your program. The line number specified is the first line of a subroutine to which control is transferred when the GOSUB is encountered.

GOSUB is different from the GOTO statement in that the system stores the location of the GOSUB statement and returns control to the statement following the GOSUB when the subroutine is finished executing.

EXAMPLE:

```
>listnh <ret>
10 PRINT "Enter two numbers separated by a comma";
20 INPUT x,y
30 GOSUB 110
40 PRINT
50 PRINT "The subroutine just completed."
60 PRINT
70 PRINT "1) x + y = ";a
80 PRINT
90 PRINT "2) x * y = ";b
100 GO TO 170
110       REM This is the subroutine
120       PRINT "The subroutine just started."
130       a = x + y
140       WAIT(2)
150       b = x * y
160  RETURN
170 END
```

```
>runnh <ret>

Enter two numbers separated by a comma? 23.4,58.2 <ret>
The subroutine just started.

The subroutine just completed.

1) x + y =  81.6

2) x * y =  1361.88

>
```

Statement
GOTO

GOTO Statement

SYNTAX: GOTO line number

DESCRIPTION:

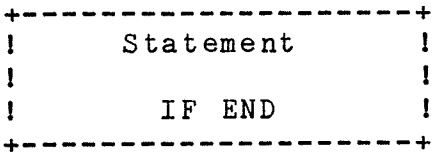
The GOTO statement is an unconditional branching statement that transfers control to the specified line number and resumes execution at that point. Unlike GOSUB, the system does not store the location of the GOTO statement and does not return control to that point.

The target line number in a GOTO statement can be anywhere in the program. If the line number does not exist in the program, a warning message is returned.

EXAMPLE:

```
>listnh <ret>
10 gamma = 5
20 PRINT gamma * 10
30 GO TO 60
40 PRINT "This line will not execute in this program"
50 PRINT gamma * 3
60 PRINT
70 PRINT "Skip to line 60."
80 PRINT
90 PRINT gamma * 15
100 END
```

```
>runnh <ret>
 50
Skip to line 60.
 75
>
```



IF END Statement

SYNTAX: IF END #expression THEN statement

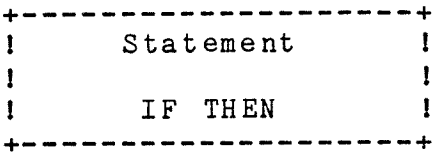
or

IF END #expression THEN line number

DESCRIPTION:

The IF END conditional statement is used to test for an end of file condition. It is used with the INPUT # statement to determine if there is no more data to read from a file. The statement is similar in function to the IF-THEN statement.

In the syntax above expression is the channel number of the file to test for, statement is any valid I/OBASIC statement, and line number is any valid I/OBASIC program line number. See the chapter Using Data Files in the BASYS User's Guide for more information on the IF END statement.



IF THEN Statement

SYNTAX: IF condition THEN statement

or

IF condition THEN line number

DESCRIPTION:

The IF-THEN conditional statement is used to test for a specified condition and perform an action only if that condition is met. In I/OBASIC the condition is a relational expression between two operands. These two operands may be strings, alpha variables, constants, numeric variables, integer variables, or expressions.

When the IF-THEN statement is encountered, the system tests the condition. If it is true, the THEN statement is executed. If the condition statement is not true, the THEN statement is ignored and the next sequential statement in the program is executed.

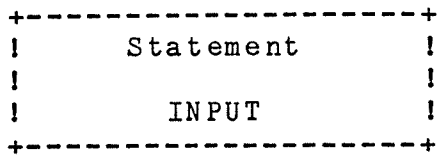
The following operators may be used in a conditional statement:

<	Less than
<=	Less than or equal
=	Equal
>=	Greater than or equal
>	Greater than
<>	Not equal to

EXAMPLE:

```
>listnh <ret>
10 PRINT "Enter 2 words";
20 INPUT word1$,word2$
30 PRINT
40 IF word1$ = word2$ THEN PRINT "Words are equal"
50 PRINT
60 IF word1$ <> word2$ THEN PRINT "Words not equal"
70 PRINT
80 IF word1$ <> word2$ THEN 10
90 END

>runnh <ret>

Enter 2 words? house, mouse <ret>

Words not equal

Enter 2 words? house, house <ret>

Words are equal
```

```
>old prog6 <ret>

>listnh <ret>
10 PRINT "Enter 2 numbers (0 to end)";
20 INPUT number1,number2
30   IF number1 = 0 THEN 110
40   IF number2 = 0 THEN 110
50   IF number1 <> number2 THEN PRINT "Not Equal"
60   IF number1 = number2 THEN PRINT "Equal"
70 PRINT
100 GO TO 10
110 PRINT "That is the end."

>runnh <ret>

Enter 2 numbers (0 to end)? 23.4,56 <ret>
Not Equal

Enter 2 numbers (0 to end)? 55.4,55.4 <ret>
Equal

Enter 2 numbers (0 to end)? 23.6,0 <ret>
That is the end.

>
```



INPUT Statement

SYNTAX: INPUT variable1 [,variable2 ...]

DESCRIPTION:

The INPUT statement allows the user to interact directly with the program to enter values. When an INPUT statement is executed, the program halts and prints a prompt on the terminal screen. If more than one variable is input, the variables should be separated by commas.

The values input must agree with the variable types specified in the program or an error will result. Note the error message returned in example 2 following input of the wrong variable type.

EXAMPLE:

```
>listnh <ret>
10 message$ = "Enter your full name"
20 PRINT message$;
30 INPUT fullname$
40 PRINT "Hello ";fullname$
50 END

>runnh <ret>

Enter your full name? John Jones <ret>
Hello John Jones

>old prog21 <ret>

>listnh <ret>
10 prompt$ = "Enter your age to the nearest year"
20 PRINT prompt$
30 INPUT age%
40 PRINT "Good"
50 END

>runnh <ret>

Enter your age to the nearest year
? 25.5 <ret>
?Bad data-retype from error at line 30
25 <ret>
Good

>
```

Statement
INPUT #

INPUT # Statement

SYNTAX: INPUT #channel, variable1 [,variable2 ...]

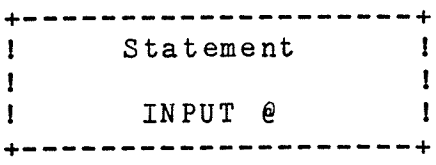
DESCRIPTION:

The INPUT # statement reads data from a file that has been opened for input, and assigns values to variable(s) specified in the INPUT statement. If the line of data in the file contains more data than the number of variables in the INPUT statement, the excess data is ignored. If the channel number specified is zero, the values are input from the user's terminal, but no prompt is output.

The INPUT # statement is not supported in MICROBASYS or PICOBASYS.

EXAMPLE:

```
>listnh <ret>
10  OPEN "REC.DAT" FOR INPUT AS FILE #1
20   FOR i = 1 TO 365
30     INPUT #1, daily_receipts
40     total_receipts = daily_receipts + total_receipts
50   NEXT i
60  PRINT "Total Receipts for Year = ";total_receipts
70  CLOSE #1
80  END

>runnh <ret>

Total Receipts for Year = 332650.00

>
```



INPUT @ Statement

SYNTAX: INPUT @channel, variable1 [,variable2 ...]

DESCRIPTION:

The INPUT @ statement reads data from a serial channel. It acts like the standard INPUT statement, except that input is obtained from the serial channel specified in the INPUT @ statement. The program waits for all data to be input while it is executing an INPUT @ statement. Unlike the standard INPUT statement, no prompt is issued at the serial channel.

Characters typed at a serial channel are buffered so that they are not lost, even if an INPUT @ statement is not being executed. The characters typed are always echoed after an INPUT @ statement executes. Note that the CHAR_IN statement will disable echoing at a serial channel until an INPUT @ or LINPUT @ statement is again executed.

The console terminal is always serial channel number zero. Legal values for the channel number can range from 0 to 7 for DX11 and PX11.

EXAMPLE:

```
>listnh <ret>
10 REM get input from several serial channels
20 FOR term_no = 1 to 3
30   PRINT @term_no, "Enter a number please: "
40   INPUT @term_no, any_number
50 NEXT term_no
60 END

>
```

Function
INT

INT Function

SYNTAX: INT(numeric expression)

DESCRIPTION:

The INT function returns the greatest integer less than or equal to the numeric expression supplied.

By using the formula below, the INT function can be made to round any number N to D decimal places:

INT(N*10^D+0.5)/10^D

EXAMPLE:

```
>listnh <ret>
10 FOR series = 1 TO 3
20       PRINT "Enter a number";
30       INPUT number
40       PRINT "The integer of ";number;" = ";INT(number)
50       PRINT
60 NEXT series
70 END
```

```
>runnh <ret>

Enter a number? 2.25 <ret>
The integer of  2.25  =  2

Enter a number? 3.999 <ret>
The integer of  3.999  =  3

Enter a number? 999 <ret>
The integer of  999  =  999

>
```

Statement
KILL

KILL Statement

SYNTAX: KILL string

DESCRIPTION:

The KILL statement deletes the file specified by the string. Once a KILL statement has been executed, the deleted file can no longer be opened or any of the records in the file accessed since the file is permanently deleted.

Function
LEN

LEN Function

SYNTAX: LEN(string expression)

DESCRIPTION:

The LEN function is used to determine the number of characters in the specified string. The LEN function returns an integer equal to the number of characters in the specified string including leading and trailing blanks. For example, LEN(A$) will return the number of characters in the string A$.

EXAMPLE:

```
>listnh <ret>
10 b$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 PRINT "The length of the string is ";LEN(b$)
30 END

>runnh <ret>

The length of the string is  26

>
```

Command
LENGTH

LENGTH Command

SYNTAX: LENGTH

DESCRIPTION:

The LENGTH command returns the size of the current program in memory, and the size of the free memory available for additional program space. The size is given in bytes. This size can change when a program is run depending upon how much dynamic memory is needed for arrays, string variables, and open files.

Statement
LET

LET Statement

SYNTAX: [LET] variable = expression

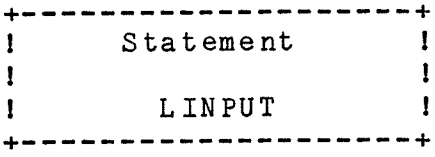
DESCRIPTION:

The LET statement assigns the value of the expression on the right of the equal sign to the variable on the left. The expression may be a string, integer, or number, but the data type of the expression and variable must either both be strings or numeric. That is, a string expression cannot be assigned to a numeric variable.

Strings should be enclosed in quotation marks. The LET portion of the statement is optional.

EXAMPLE:

```
>listnh <ret>
10 LET alpha = 5
20 PRINT "The first value of alpha = ";alpha
30 LET alpha = alpha + 1
40 PRINT "The second value of alpha = ";alpha
50 alpha = 10
60 PRINT "The final value of alpha = ";alpha
70 END

>runnh <ret>

The first value of alpha =  5
The second value of alpha =  6
The final value of alpha =  10

>
```



LINPUT Statement

SYNTAX: LINPUT string variable1 [,string variable2 ...]

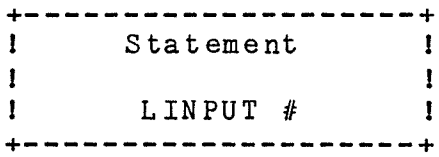
DESCRIPTION:

The LINPUT (Line Input) statement is used exclusively to enter string variables directly from the console serial channel. Unlike the INPUT statement which interprets the quotation marks and commas as delimiters, the LINPUT statement accepts entire strings. This means that multiple LINPUT responses cannot be entered on the same line. String data entered in response to a LINPUT statement must be entered individually followed by a carriage return.

EXAMPLE:

```
>listnh <ret>
10    prompt1$ = "last name"
20    prompt2$ = "first name"
30  PRINT "Enter ";prompt1$
40 LINPUT lastname$
50  PRINT "Enter ";prompt2$
60 LINPUT firstname$
70  PRINT
80  PRINT "My  name is ";firstname$;" ";lastname$
```

```
>runnh <ret>

Enter last name
? Bailey, 3rd<ret>
Enter first name
? William "Bill" <ret>

My name is William "Bill" Bailey, 3rd

>
```

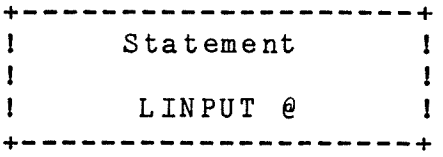


LINPUT # Statement

SYNTAX: LINPUT #channel ,string variable1 [,string variable2 ...]

DESCRIPTION:

The LINPUT # statement is used for reading strings from a data file. It acts like the INPUT # statement in that it obtains input from a file. The LINPUT # statement accepts entire strings as input, ignoring comma and quotation mark delimiters. See the LINPUT statement for more information.



LINPUT @ Statement

SYNTAX: LINPUT @channel, string variable1 [,string variable2 ...]

DESCRIPTION:

The LINPUT @ statement reads data from a serial channel. It acts like the standard LINPUT statement, except that input is obtained from the serial channel specified in the LINPUT @ statement. The program waits for all data to be input while it is executing a LINPUT @ statement. Unlike the standard LINPUT statement, no prompt is issued at the serial channel.

Characters typed at a serial channel are buffered so that they are not lost, even if a LINPUT @ statement is not being executed. The characters typed are always echoed after a LINPUT @ statement executes. Note that the CHAR_IN statement will disable echoing at a serial channel until an INPUT @ or LINPUT @ statement is again executed.

The console serial channel is always channel number zero. Legal values for the channel number can range from 0 to 7 for DX11 and PX11.

Command
LIST/LISTNH

LIST/LISTNH Command

SYNTAX: LIST[NH] [line specification1 ...]

DESCRIPTION:

The LIST command instructs the system to display the text currently in memory. If no line number(s) is specified, the entire program will be displayed along with a header which identifies the file name, date and time of day.

The LISTNH command is identical to the LIST command, but a header is not displayed with the program text. The header contains the current date, time and program name. The date will not be listed in the header if no date has been entered into the system.

The SAVE command should be used to list a program on a printer or device other than a terminal.

EXAMPLE:

>list <ret>	<The entire program will be <displayed on the terminal, <including a header line.
>listnh <ret>	<The entire program will be <displayed on the terminal, with <no header line.
>list 20,30 <ret>	<Lines 20 and 30 are displayed.
>list 20-50 <ret>	<Lines 20 through 50 are <displayed.
>list 10,30-60 <ret>	<Lines 10 and 30 through 60 are <displayed.

Function
LOG

LOG Function

SYNTAX: LOG(numeric expression)

DESCRIPTION:

The LOG function returns the logarithm to the base e of the expression specified. The value of e is approximately 2.71828. The natural logarithm of a number X is the power to which e must be raised to equal X. For example, the natural logarithm of 100 is 4.60517, because e raised to the power 4.60517 equals 100.

The LOG function is the inverse of the EXP function.

If the value of the expression specified is negative or zero, I/OBASIC prints the nonfatal error message, "?Bad log" and returns a value of 0.

Note that execution of the sample program is terminated by typing a CTRL/C.

EXAMPLE:

```
>listnh <ret>
10    PRINT "Enter a number";
20 INPUT beta
30    PRINT "The natural logarithm of ";beta;
40    PRINT " is ";LOG(beta)
50    PRINT
60 GO TO 10

>runnh <ret>

Enter a number? 100 <ret>
The natural logarithm of  100  is  4.60517

Enter a number? 2.71829 <ret>
The natural logarithm of  2.71829  is  1

Enter a number? 45 <ret>
The natural logarithm of  45  is  3.80666

Enter a number? ^C
Stop at line 20

>
```

Function
LOG10

LOG10 Function

SYNTAX: LOG10(numeric expression)

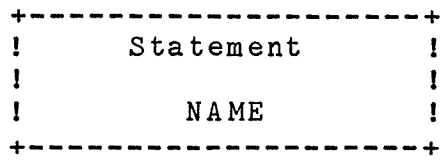
DESCRIPTION:

The LOG10 function returns the logarithm to the base 10 of the expression specified. The logarithm to the base 10 of N is the power to which 10 must be raised to equal N. For example, the logarithm to base 10 of 100 is 2 because 10 to the power 2 equals 100.

If the value of the expression specified is negative or zero, I/OBASIC prints the nonfatal error message "?Bad log" and returns a value of 0.

EXAMPLE:

```
>listnh <ret>
10 FOR range = 1 TO 100 STEP 10
20     PRINT LOG10(range)
30 NEXT range
40 END

>runnh <ret>

 0
 1.04139
 1.32222
 1.49136
 1.61278
 1.70757
 1.78533
 1.85126
 1.90849
 1.95904

>
```



NAME Statement

SYNTAX: NAME string1 TO string2

DESCRIPTION:

The NAME statement renames the file specified by string1, giving it the name specified by string2. If a device is specified, it must be the same in string1 and string2.

When the NAME statement is executed, only the name of the file is changed; the contents remain unaltered.

NEW Command

SYNTAX: NEW [program name]

DESCRIPTION:

The NEW command clears all program text presently in immediate memory and allows the user to enter a new program.

In DISKBASYS and PROMBASYS, if a program name is not provided in the command, I/OBASIC will request the name of the program to be created. If you press the carriage return without entering a program name then the new program is given the default name "NONAME."

EXAMPLE:

```
>new prog1 <ret>          <The system clears all immediate
                          <memory and names the new program
                          <to be entered, PROG1.

>new <ret>
New file name?            <System prompts for name if none
                          <is supplied with the command.
                          <If no name is supplied in
                          <response to this prompt, the
                          <system assigns the name NONAME.
```

Statement
NEXT

NEXT Statement

SYNTAX: NEXT numeric variable

DESCRIPTION:

The NEXT statement is always used following a FOR-TO statement. The NEXT statement terminates the FOR-TO-NEXT loop. FOR and NEXT must specify the same variable counter. See FOR-TO-NEXT.

EXAMPLE:

```
>listnh <ret>
10 PRINT "Graph of Count"
20 FOR count = 1 TO 8
30   PRINT TAB(count + 10);"*"
40 NEXT count
50 PRINT "End of Count"
60 END

>runnh <ret>

Graph of Count
           *
            *
             *
              *
               *
                *
                 *
                  *
End of Count

>
```

Function
OCT

OCT Function

SYNTAX: OCT(octal number string)

DESCRIPTION:

The OCT function returns the decimal equivalent of the octal number specified. This function converts a string representing an octal value to its numeric equivalent.

The only characters permitted in the string are 0 through 7 and space. Spaces in the expression are ignored and the value is returned as an integer. If conversion of the string to octal would result in a value outside the range -32768 to +32767, an I/OBASIC error message will be returned.

EXAMPLE:

```
>print oct('177777') <ret>
-1

>print oct('77777') <ret>
32767

>print oct("100000") <ret>
-32768

>
```

Command
OLD

OLD Command

SYNTAX: OLD [file specification]

DESCRIPTION:

The OLD command searches for the file specified, clears the immediate memory, all variables, arrays and functions, and loads the program into immediate memory. If a file specification is not specified in the command line, I/OBASIC will prompt for the file name.

If the specified file is not found, I/OBASIC returns an error message.

Statements cleared from immediate memory cannot be restored unless they have been saved using the SAVE or REPLACE commands.

EXAMPLE:

```
>old test1 <ret>          <The immediate memory will be
                          <cleared and program TEST1 will
                          <be loaded for execution or
                          <editing.

>old <ret>                <If no file specification is
Old file name -           <provided, the system will prompt
                          <you to provide one.
```

Statement
ON ERROR GOTO

ON ERROR GOTO Statement

SYNTAX: ON ERROR GOTO line number

DESCRIPTION:

The ON ERROR GOTO statement enables error processing within an I/OBASIC program. When error processing is enabled and a program error occurs (such as division by zero) the program will branch to the line number specified in the ON ERROR GOTO statement. Error processing enables you to correct for some fatal and nonfatal error conditions that could occur within a program.

When error processing is desired in an I/OBASIC program, the ON ERROR GOTO statement is generally placed at the beginning of the program. Specifying a line number of zero will disable error processing. By default, error processing is disabled when a program is run.

The error processing routine that starts at the line number given in the ON ERROR GOTO statement can make use of the ERL and ERR functions to determine the type and location of a program error. In general, the error processing routine is specific to a given program, and it is written to process only certain types of errors that would be likely to occur because of the programs particular application (such a frequent numeric overflows during calculations).

The RESUME statement is used to exit the error processing routine and either retry the program statement that caused the error or resume execution at another line number.

Statement
ON EVENT GOSUB

ON EVENT GOSUB Statement

SYNTAX: ON EVENT GOSUB line number

DESCRIPTION:

The ON EVENT GOSUB statement enables event processing within an I/OBASIC program. When event processing is enabled, and a flag argument becomes set in a real-time control statement, the program will branch to the line number specified in the ON EVENT GOSUB statement

When event processing is desired in an I/OBASIC program, the ON EVENT GOSUB statement is generally placed at the beginning of the program. Specifying a line number of zero will disable event processing. By default, event processing is disabled when a program is run.

Event processing is very useful for creating powerful real-time control programs that do not have to poll for completion of I/O or timed events.

Only one event processing subroutine can be specified by an ON EVENT GOSUB statement. The last ON EVENT GOSUB statement executed determines the line number of the subroutine.

Although there is only one event processing subroutine possible within an I/OBASIC program, the event processing subroutine can determine what flag caused the event by checking to see if a flag is zero or non-zero.

The following real-time control statements support the flag arguments and therefore can be used with the ON EVENT GOSUB statement:

ANALOG_IN
ANALOG_LOW_IN
ANALOG_OUT
CLOCK_OUT
DIGITAL_IN
DIGITAL_OUT
TEMPERATURE_IN
TIME_OUT

NOTE

It is important to note that control will branch to the event processing subroutine when a flag is set only after the current I/OBASIC statement completes entirely. For this reason, statements such as INPUT should be avoided in a program if fast real-time response is desired. The INPUT statement can theoretically take forever to execute if it is waiting for terminal input. The CHAR_IN statement could be used in place of the INPUT statement for this purpose.

EXAMPLE:

```
10 REM Check for high voltage every 0.5 seconds
20 ON EVENT GOSUB 1000
30 TIME_OUT(0,0.5,time_flag)
        .
        .
  perform other processing here
        .
        .
1000 REM event processing subroutine starts here
1010   TIME_OUT(0,0.5,time_flag) \ REM restart timer
1020   ANALOG_IN(0,voltage)
1030   IF voltage > high_limit THEN PRINT "High Voltage"
1040 EVENT RETURN
        .
        .
  other program statements
        .
        .
2000 END
```

Statement
ON GOSUB

ON GOSUB Statement

SYNTAX: ON expression GOSUB line number1 [,line number2 ...]

DESCRIPTION:

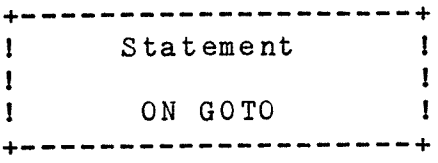
The ON GOSUB statement branches the program to the subroutine specified depending on the value of the expression. After the program has branched to the subroutine, it will return to the line following the ON GOSUB statement when it encounters a RETURN statement.

When I/OBASIC executes the ON GOSUB statement, it evaluates the expression and truncates it to an integer if necessary. If the value of the expression is 1, I/OBASIC transfers control to first line number specified; if the value of the expression is 2, I/OBASIC transfers control to the second line number specified. If the expression is less than one or greater than the number of lines specified, an I/OBASIC error message is returned.

EXAMPLE:

```
>listnh <ret>
10 PRINT "Enter Pay Code (1=Reg Time, 2=Overtime)";
20   INPUT timecode
30 PRINT "Enter Payrate";
40   INPUT rate
50 PRINT "Enter Employee Name";
60   INPUT employee$
70 PRINT "Enter number of hours worked";
80   INPUT hours
90 ON timecode GOSUB 150,190
100 PRINT "Continue";
110   INPUT a$
120 IF a$ = "YES" THEN 10
130 IF a$ = "NO" THEN 230
140 GO TO 100
150       REM Regular Time
160       pay = rate * hours
170       PRINT "Employee: ";employee$;" Regular: $";pay
180 RETURN
190       REM Overtime
200       pay = (rate * hours) * 1.5
210       PRINT "Employee: ";employee$;" Overtime: $";pay
220 RETURN
230 END
```

```
>runnh <ret>

Enter Pay Code (1=Reg Time, 2=Overtime)? 1 <ret>
Enter Payrate? 5.00 <ret>

Enter Employee Name? George Roberts <ret>
Enter number of hours worked? 10 <ret>

Employee: George Roberts Regular: $ 50
Continue? YES <ret>

Enter Pay Code (1=Reg Time, 2=Overtime)? 2 <ret>
Enter Payrate? 5.00 <ret>

Enter Employee Name? George Roberts <ret>
Enter number of hours worked? 10 <ret>

Employee: George Roberts Overtime: $ 75
Continue? NO <ret>

>
```



ON GOTO Statement

SYNTAX: ON expression GOTO line number1 [,line number2 ...]

DESCRIPTION:

The ON GOTO statement pair is a conditional branching statement. Control transfers to the line number specified in order as they correspond to the value of the expression specified. Transfer depends on the value of the expression specified.

When I/OBASIC executes the ON GOTO statement, it evaluates the expression and truncates it to an integer if necessary. If the value of the expression is 1, I/OBASIC transfers control to the first line number specified; if the value of the expression is 2, I/OBASIC transfers control to the second line number specified, etc. If the value of the expression is less than 1 or greater than the number of lines specified, an I/OBASIC error message is returned.

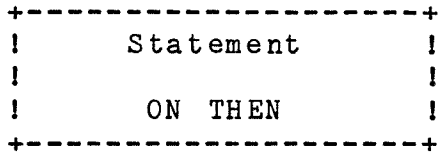
EXAMPLE:

```
>10 ON count GO TO 40,50,60,70
```

In this example, if count = 1, the program will branch to line 40, if count = 2, the program will branch to line 50, if count = 3, the program will branch to line 60, and if count = 4 the program will branch to line 70.

If the value of count is not an integer, the fractional part will be ignored. If count is negative or larger than the number of line numbers specified (in this example 5 or greater), an error message will be returned.

Unlike the ON GOSUB statement, when the program branches to the line number specified, it does not return to the line following the ON GOTO statement.

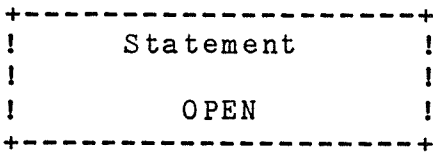


ON THEN Statement

SYNTAX: ON expression THEN statement

DESCRIPTION:

The ON-THEN statement is identical to the ON-GOTO statement.



OPEN Statement

SYNTAX: OPEN string [FOR INPUT] AS FILE [#]expr

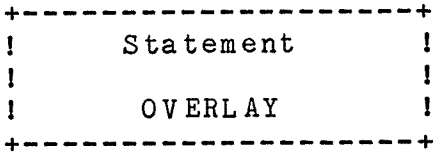
or

OPEN string [FOR OUTPUT] AS FILE [#]expr

DESCRIPTION:

The OPEN statement associates a file channel number with a file specification. The OPEN statement is used to either open an existing file or create a new file. It must be executed before any file I/O can take place. In the syntax above, string is any valid file specification, and expr is the file channel number to associate with the file. The file channel number can be any integer in the range 1 to 12.

See the chapter Using Data Files in the BASYS User's Guide for more information on the OPEN statement.



OVERLAY Statement

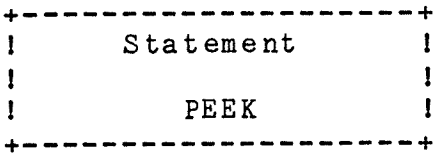
SYNTAX: OVERLAY file specification [line number]

DESCRIPTION:

The OVERLAY statement merges one program with another in immediate memory by overlaying the new on the old. All common lines are replaced by the new program lines. All lines not common to the two programs remain unchanged.

When the OVERLAY statement is executed, the program specified is merged with the program currently in immediate memory and execution begins with the program line number specified. If no beginning line number is specified, execution starts at the lowest line number in immediate memory after the two programs are merged.

If a beginning line number is specified that does not exist, I/OBASIC returns the error message, "?Undefined line number".



PEEK Statement

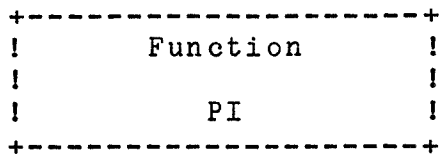
SYNTAX: PEEK(address, val)

DESCRIPTION:

The PEEK statement will read a 16-bit word from a memory location.

Argument address is a variable or constant of any type that contains the memory address that will be read. The value of this argument must be supplied by the program, and it must be an even number. Legal range is 0 to 177776 (octal). If the memory location is not present, then an error message will be printed.

Argument val is a variable of any type that will be returned with the contents of the word at the memory location specified by argument address.



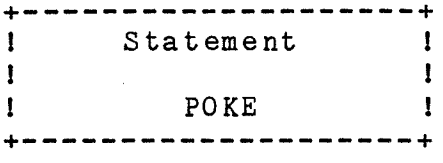
PI Function

SYNTAX: PI

DESCRIPTION:

The PI function returns the value of PI (approximately 3.141593). Pi is the ratio of the circumference of a circle to its diameter.

This function needs no argument; it can be used in any numeric expression.



POKE Statement

SYNTAX: POKE(address, val)

DESCRIPTION:

The POKE statement will write a 16-bit word to a memory location.

Argument address is a variable or constant of any type that contains the memory address that will be written to. The value of this argument must be supplied by the program, and it must be an even number. Legal range is 0 to 177776 (octal). If this address is not present, then an error message will be printed.

Argument val is a variable or constant of any type that contains the word to write to the memory location specified by argument address. The value of this argument must be supplied by the program. Legal range is 0 to 177777 (octal).

```
+--------------------+
!      Function      !
!                    !
!        POS         !
+--------------------+
```

POS Function

```
                      string to              position to
SYNTAX:         POS(be searched,substring,begin search)
```

DESCRIPTION:

The POS (Position) function will search for a specified substring within the string specified for its location. I/OBASIC begins the search at the character position specified. If the substring is located, the integer value of the location of the first character in the substring is returned. If the substring is not found, a zero is returned. The count is from left to right within the string to be searched. Spaces within the string are not ignored.

EXAMPLE:

```
>listnh <ret>
10 a$ = "The quick brown fox jumped over the lazy dog"
20 b$ = "fox jumped"
30 PRINT "The string starts at position ";POS(a$,b$,1)
40 END

>runnh <ret>

The string starts at position 17

>
```

```
+--------------------+
!     Statement      !
!                    !
!       PRINT        !
+--------------------+
```

PRINT Statement

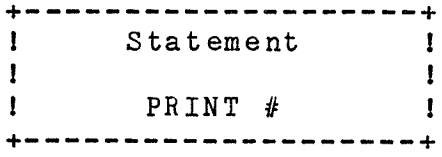
SYNTAX: PRINT [list]

DESCRIPTION:

The PRINT statement outputs data to the console terminal.

The optional list argument contains all items to be printed and can contain any numeric, string or tab function. The items in the list argument may be separated by commas or semicolons.

Comma separators will cause the list items to be printed in different zones. Semicolon separators will cause the list item to be printed in packed format. If no list is specified, a blank line is printed. If a semicolon is the last character in the list a carriage return and line feed will not be sent to the console terminal after the PRINT statement executes.



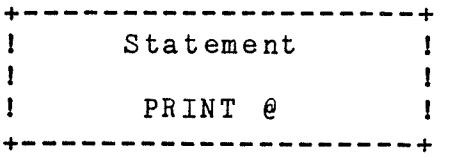
PRINT # Statement

SYNTAX: PRINT #channel [,list]

DESCRIPTION:

The PRINT # statement outputs data to the file associated with the specified channel. If 0 is specified, or if the channel number is omitted, then the data is output to the user's terminal.

The optional list argument contains all items to be printed and can contain any numeric, string or tab function. The items in the list argument may be separated by commas or semicolons. Comma separators will cause the list items to be printed in different zones. Semicolon separators will cause the list item to be printed in packed format. If no list is specified, a blank line is printed.



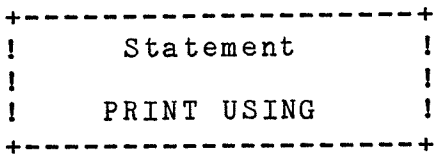
PRINT @ Statement

SYNTAX: PRINT @channel [,list]

DESCRIPTION:

The PRINT @ statement outputs data to a serial channel. It acts like the standard PRINT statement, except that output is directed to the serial channel specified in the PRINT @ statement.

The console terminal is always serial channel number zero. Legal values for the channel number can range from 0 to 7 for DX11 and PX11.



PRINT USING Statement

SYNTAX: PRINT USING string, list

or

PRINT #channel, USING string, list

or

PRINT @channel, USING string, list

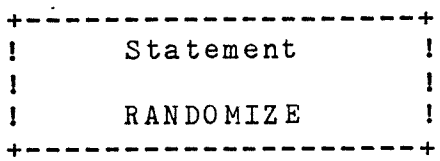
DESCRIPTION:

The PRINT USING statement is a variation of the PRINT statement that can be used to format the output sent to a terminal or a file.

For numbers, the PRINT USING statement can control the format of the number of digits, location of the decimal point, inclusion of special symbols, and exponential format.

For strings, the PRINT USING statement can control the format of the number of characters printed and justification.

A full description of the PRINT USING statement is provided in the chapter Formatted Printing in the BASYS User's Guide.



RANDOMIZE Statement

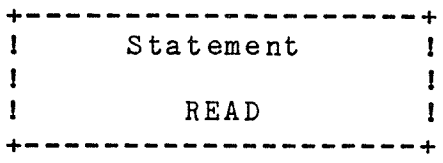
SYNTAX: RANDOMIZE

DESCRIPTION:

The RANDOMIZE statement is used in conjunction with the RND function. Every time I/OBASIC executes the RANDOMIZE statement it begins the RND function at a new and unpredictable location in the series. This allows an apparently unrelated and unpredictable number to be produced at any time.

NOTE

Do not include the RANDOMIZE statement until you have debugged your program. If you do, you will not know if changes to your program or the RANDOMIZE statement are producing changing results.



READ Statement

SYNTAX: READ variable1 [,variable2 ...]

DESCRIPTION:

The READ statement is always used with the DATA statement. It inputs values from the DATA statement. For each variable specified in a READ statement, I/OBASIC retrieves the next value in a DATA statement. The DATA statement provides the values and the READ statement assigns those values to the specified variables.

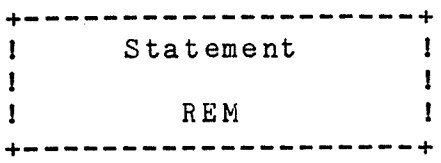
A data pointer keeps track of the data read. Values are assigned in sequence from the DATA statement each time the READ statement is executed. READ and DATA statements are generally used to reinitialize variables with known values.

READ statements can appear anywhere in a multi-statement line, but are not valid unless there is at least one DATA statement in the program.

Variables and values may be either numeric or string and both types of variables and values may occur in the same READ/DATA statement pair. The numbers of each type of variable and value must be the same in the DATA statement and the READ statement.

EXAMPLE:

```
>10   READ a,b,c
>20   DATA  6,1000,-0.087


>10   READ any_array(pointer)
>80   DATA  2,6,890,-19,54


>10   READ a$,b$,c$
>240  DATA  Time,Hours,Pay
>250  DATA Time, Overtime, Doubletime


>10   READ alpha$,alpha,beta$,beta,gamma$,gamma
>350  DATA  DEGREES,65,VOLTS,210,HOURS,1
```



REM Statement

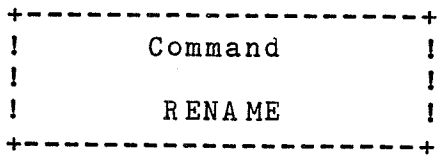
SYNTAX: REM comment

DESCRIPTION:

The REM statement is used to insert comments into a program to identify or explain features of the program. The system ignores any statement beginning with the REM statement.

EXAMPLE:

```
>10 REM Heat Conversion Program
>20 REM Revision 2/4 April 1982


>250 REM Data Collection Subroutine Starts Here
```



RENAME Command

SYNTAX: RENAME [new name]

DESCRIPTION:

The RENAME command gives the specified new name to the current program in memory. The RENAME command will not change the name of a file, or alter the contents of a file.

A program name can contain up to six letters and/or numbers. If the new name is not specified with the RENAME command, I/OBASIC will prompt for the new name.

REPLACE Command

SYNTAX: REPLACE [file specification]

DESCRIPTION:

The REPLACE command performs the same function as the SAVE command, except that any existing file with the same file specification is deleted and replaced with the program in immediate memory. If no file is specified in the command, the name of the program currently in immediate memory is used.

The REPLACE command is used to update a stored version of a file with a newer version after editing it in immediate memory.

EXAMPLE:

```
>replace test1 <ret>     <The stored program TEST1
                         <is replaced with the new
                         <version in immediate memory.

>replace <ret>           <The program in immediate
                         <memory is saved and any
                         <stored program with the
                         <same name is replaced.
```

Command
RESEQ

RESEQ Command

SYNTAX: RESEQ [new line number],[old line number1]-[old line number2],[increment]

DESCRIPTION:

The RESEQ command resequences the line numbering of the program in immediate memory. New line number specifies the starting number of the new sequence. If you specify 50, the program will be renumbered starting with line number 50. Old line number1 - old line number2 specifies the segment of the program to be renumbered. If these parameters are not specified, the entire program is renumbered. Increment specifies the increment to be used in the new numbering.

Before running the program, user should SAVE resequenced program first. Then, use OLD command to retrieve program before running.

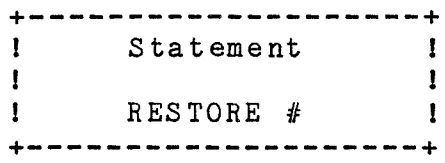
Statement
RESTORE

RESTORE Statement

SYNTAX: RESTORE

DESCRIPTION:

The RESTORE statement resets the READ-DATA pointer to the beginning of the first data value. This allows reuse of data values and avoids out of data errors. The RESTORE statement can be executed as many times as necessary. It is useful for rereading of the data in DATA statements.



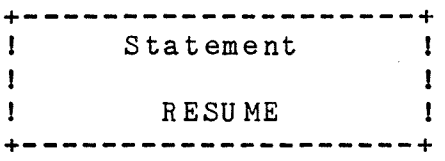
RESTORE # Statement

SYNTAX: RESTORE #channel

DESCRIPTION:

The RESTORE # statement will reset the data pointer for a specified input file to the beginning. In the syntax above channel is the channel number of the file to be restored.

This statement is useful for reading a file a multiple number of times, since the file does not have to be closed and reopened each time. The statement operates in a similar manner to the RESTORE statement that is used to reset the READ pointer for data contained in DATA statements.



RESUME Statement

SYNTAX: RESUME [line number]

DESCRIPTION:

The RESUME statement will resume normal program execution after an error has occured and the program has branched to the line number specified in an ON ERROR GOTO statement. The RESUME statement is the last statement in an error processing subroutine. It re-enables error processing after an error occurs within a program.

If a line number is not specified with the RESUME statement, then I/OBASIC will branch to and re-execute the entire line that caused the program error. If a line number is specified, then RESUME will act as a GOTO statement, except that error processing will be re-enabled.

See descriptions of the ERL and ERR functions and the ON ERROR GOTO statement for additional information about error processing.

RETURN Statement

SYNTAX: RETURN

DESCRIPTION:

The RETURN statement terminates a subroutine and returns control to the statement following the last executed GOSUB statement. If a RETURN statement is encountered and a GOSUB has not been executed, an error message is returned, "?RETURN without GOSUB".

EXAMPLE:

```
>listnh <ret>
10  REM Example of a RETURN
20  PRINT "Enter a number"
30  number2 = 5
40  INPUT number1
50  GOSUB 100
60  PRINT "The subroutine just finished."
70  PRINT "The program is ended."
80  GO TO 130
90  REM Subroutine starts here
100 PRINT "Number 1:";number1
110 PRINT "Number 2:";number2
120 RETURN

>runnh <ret>

Enter a number
? 7 <ret>

Number 1: 7
Number 2: 5
The subroutine just finished.
The program is ended.

>
```

```
+--------------------+
!      Function      !
!                    !
!        RND         !
+--------------------+
```

RND Function

SYNTAX: RND

DESCRIPTION:

The RND function returns a pseudo-random number between zero and one. I/OBASIC maintains a series of apparently unrelated numbers called a pseudo-random series. Each time the RND function is used, the next number in the series is returned.

Whenever the program is initialized through the use of an OLD, NEW, or RUN command, or a CHAIN statement, the random series pointer is set back to the beginning of the series. Therefore, a predictable set of random numbers is always available for testing. When the RANDOMIZE statement is used with the RND function, the pointer is set to an unpredictable location and the series is returned from that point on.

EXAMPLE:

```
>listnh <ret>
10 REM Example of RND function
20 PRINT RND, RND, RND
30 END

>runnh <ret>
.0407319        .528293         .803172

>runnh <ret>
.0407319        .528293         .803172

>old prog11 <ret>

>listnh <ret>
10 REM Example of RND function with RANDOMIZE
20 RANDOMIZE
30 PRINT RND, RND, RND
40 END

>runnh <ret>
.494162         .888584         .884043

>runnh <ret>
.528293         .803172         .0643915

>
```

Command
RUN/RUNNH

RUN/RUNNH Command

SYNTAX: RUN[NH] [file specification]

DESCRIPTION:

The RUN command is used to begin program execution. When the RUN command is entered, a header line is printed and the program in immediate memory is run.

In DISKBASYS and PROMBASYS, if a file specification is supplied, immediate memory is cleared, the program specified is loaded from mass storage into immediate memory, and execution begins. In MICROBASYS, any file specification supplied with the RUN or RUNNH command will be ignored, and the program currently in memory will be run.

The RUNNH command is identical to RUN, but printing of the header line is suppressed. The header contains the current date, time and program name. The date will not be listed in the header if no date has been entered into the system.

SAVE Command

SYNTAX: SAVE [file specification]

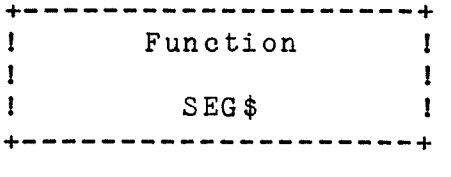
DESCRIPTION:

The SAVE command outputs the program in immediate memory to the specified file. If no file is specified, the program name in immediate memory is the default specification. If the file specified already exists in memory, an error message is returned, "?Use REPLACE".

The SAVE command can be used to save a second copy of a program under a different name. SAVE can also be used to output a program to a device, such as a line printer. This is done by specifying a device rather than a file name.

EXAMPLE:

```
>save <ret>              <The program is saved under the
                         <current name.

>save prog1 <ret>        <The program is save under the
                         <name PROG1.

>save prog1 <ret>        <Specifying an existing file name
?Use REPLACE             <results in an error message.
```



SEG$ Function

SYNTAX: SEG$(string, expression1, expression2)

DESCRIPTION:

The SEG$ function returns a segment of a string starting from the character position given by expression1 to that given by expression2. The SEG$ function, when used with the ASC function, is useful in determining the value of any character or group of characters in a specific position within a string.

EXAMPLE:

```
>listnh <ret>
10 REM Determine if product is a semiconductor
20  PRINT "ENTER  PRODUCT  ID('AAAAA')"
30 LINPUT product_id$
40  IF SEG$(product_id$,1,1)="S" THEN 70
50  PRINT "Product is not a semiconductor."
60 GO TO 20
70  PRINT "Good! It's a semiconductor."
80 END

>runnh <ret>

ENTER PRODUCT ID('AAAAA')
? R2345 <ret>
Product is not a semiconductor.
? S2345 <ret>
Good! It's a semiconductor.

>
```

Statement
SET_AIN_GAIN

SET_AIN_GAIN Statement

SYNTAX: SET_AIN_GAIN(val)

DESCRIPTION:

The SET_AIN_GAIN statement will set the hardware programmable gain on the high-level analog input device.

Argument val is a variable or constant of any type that specifies a code for the analog gain. The value of this argument must be supplied by the program. Its legal range is 0 to 3. The table below shows the analog gain that corresponds to each possible value for argument val:

val	Gain
0	8
1	4
2	2
3	1

The default value is 3, which sets the gain to 1. This means that the analog input signal is passed unchanged to the analog input device.

Statement
SET_AINL_GAIN

SET_AINL_GAIN Statement

SYNTAX: SET_AINL_GAIN(val)

DESCRIPTION:

The SET_AINL_GAIN statement will set the hardware programmable gain on the low-level analog input device.

Argument val is a variable or constant of any type that specifies a code for the analog gain. The value of this argument must be supplied by the program. Its legal range is 0 to 7. The table below shows the analog gain that corresponds to each possible value for argument val:

val	Gain
0	1000
1	500
2	200
3	100
4	20
5	10
6	5
7	1

The default value is 7, which sets the gain to 1. This means that the full-scale analog input signal range is 10.0 volts/1 or 10.0 volts.

Statement
SET_AIN_SCAN

SET_AIN_SCAN Statement

SYNTAX: SET_AIN_SCAN

DESCRIPTION:

The SET_AIN_SCAN statement enables channel scanning for the ANALOG_IN statement. When channel scanning is enabled, argument chan in statement ANALOG_IN specifies the high channel to be read, so that channels zero to chan are sequentially read and placed in successive elements of an argument val array.

The default setting for I/OBASIC is NOSCAN. The default setting is restored each time a run command is issued.

Statement
SET_AIN_NOSCAN

SET_AIN_NOSCAN Statement

SYNTAX: SET_AIN_NOSCAN

DESCRIPTION:

The SET_AIN_NOSCAN statement disables channel scanning for the ANALOG_IN statement. When channel scanning is disabled, argument chan in statement ANALOG_IN specifies the single channel to be read.

This is the default setting for I/OBASIC. The default setting is restored each time a run command is issued.

Statement
SET_AINL_SCAN

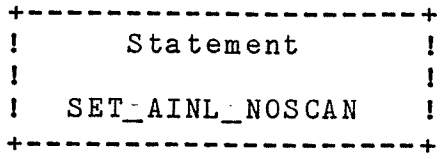
SET_AINL_SCAN Statement

SYNTAX: SET_AINL_SCAN

DESCRIPTION:

The SET_AINL_SCAN statement enables channel scanning for the ANALOG_LOW_IN statement. When channel scanning is enabled, argument chan in statement ANALOG_LOW_IN specifies the high channel to be read, so that channels zero to chan are sequentially read and placed in successive elements of an argument val array.

The default setting for I/OBASIC is NOSCAN. The default setting is restored each time a run command is issued.



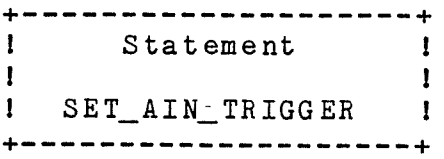
SET_AINL_NOSCAN Statement

SYNTAX: SET_AINL_NOSCAN

DESCRIPTION:

The SET_AINL_NOSCAN statement disables channel scanning for the ANALOG_LOW_IN statement. When channel scanning is disabled, argument chan in statement ANALOG_LOW_IN specifies the single channel to be read.

This is the default setting for I/OBASIC. The default setting is restored each time a run command is issued.



SET_AIN_TRIGGER Statement

SYNTAX: SET_AIN_TRIGGER

DESCRIPTION:

The SET_AIN_TRIGGER statement enables external (hardware) triggering for the ANALOG_IN statement. When external triggering is enabled, the analog input device uses either an onboard clock or a user supplied signal for triggering the A/D converter, depending on how the hardware is configured.

The default setting for I/OBASIC is NOTRIGGER. The default setting is restored when a RUN command is issued.

Statement
SET_AIN_NOTRIGGER

SET_AIN_NOTRIGGER Statement

SYNTAX: SET_AIN_NOTRIGGER

DESCRIPTION:

The SET_AIN_NOTRIGGER statement disables external (hardware) triggering for the ANALOG_IN statement. When external triggering is disabled, the analog input device is triggered by software.

This is the default setting for I/OBASIC. The default setting is restored when a RUN command is issued.

Statement
SET_AINL_TRIGGER

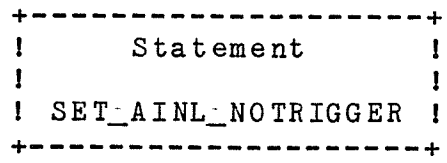
SET_AINL_TRIGGER Statement

SYNTAX: SET_AINL_TRIGGER

DESCRIPTION:

The SET_AINL_TRIGGER statement enables external (hardware) triggering for the ANALOG_LOW_IN statement. When external triggering is enabled, the analog input device uses either an onboard clock or a user supplied signal for triggering the A/D converter, depending on how the hardware is configured.

The default setting for I/OBASIC is NOTRIGGER. The default setting is restored when a RUN command is issued.



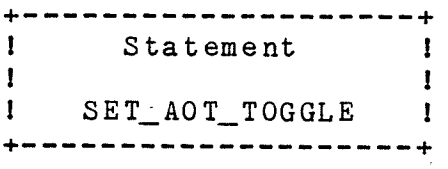
SET_AINL_NOTRIGGER Statement

SYNTAX: SET_AINL_NOTRIGGER

DESCRIPTION:

The SET_AINL_NOTRIGGER statement disables external (hardware) triggering for the ANALOG_LOW_IN statement. When external triggering is disabled, the analog input device is triggered by software.

This is the default setting for I/OBASIC. The default setting is restored when a RUN command is issued.



SET_AOT_TOGGLE Statement

SYNTAX: SET_AOT_TOGGLE

DESCRIPTION:

The SET_AOT_TOGGLE statement enables channel toggling for the ANALOG_OUT statement when using DMA mode. When channel toggling is enabled, the analog output alternates between DAC channel 1 and DAC channel 2.

The default setting for I/OBASIC is NOTOGGLE. The default setting is restored when a RUN command is issued.

The SET_AOT_TOGGLE statement is not supported in PICOBASYS.

Statement
SET_AOT_NOTOGGLE

SET_AOT_NOTOGGLE Statement

SYNTAX: SET_AOT_NOTOGGLE

DESCRIPTION:

The SET_AOT_NOTOGGLE statement disables channel toggling for the ANALOG_OUT statement when using DMA mode. DMA analog output will always occur at DAC channel 1.

This is the default setting for I/OBASIC. The default setting is restored when a RUN command is issued.

Statement
SET_AOT_TRIGGER

SET_AOT_TRIGGER Statement

SYNTAX: SET_AOT_TRIGGER

DESCRIPTION:

The SET_AOT_TRIGGER statement enables external (hardware) triggering for the ANALOG_OUT statement when it is operating in DMA mode. When external triggering is enabled, the analog output device uses either an onboard clock or a user supplied signal for triggering the D/A converter, depending on how the hardware is configured.

The default setting for I/OBASIC is NOTRIGGER. The default setting is restored when a RUN command is issued.

Statement
SET_AOT_NOTRIGGER

SET_AOT_NOTRIGGER Statement

SYNTAX: SET_AOT_NOTRIGGER

DESCRIPTION:

The SET_AOT_NOTRIGGER statement disables external (hardware) triggering for the ANALOG_OUT statement when it is operating in DMA mode. When external triggering is disabled, the analog output device is triggered by software.

This is the default setting for I/OBASIC. The default setting is restored when a RUN command is issued.

Statement
SET_ANALOG_PERCENT

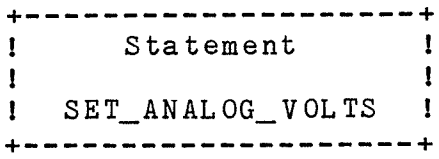
SET_ANALOG_PERCENT Statement

SYNTAX: SET_ANALOG_PERCENT

DESCRIPTION:

The SET_ANALOG_PERCENT statement sets the analog engineering units to percent full-scale, rather than volts. All values returned by analog input statements, or passed to analog output statements, are in units of percent full-scale. Unipolar signals can range from 0 to 100, and bipolar signals can range from -100 to +100.

This is the default setting for I/OBASIC. The default setting is restored when a RUN command is issued.



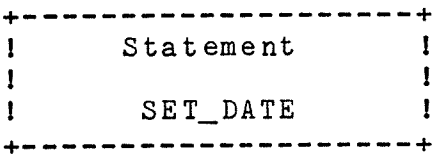
SET_ANALOG_VOLTS Statement

SYNTAX: SET_ANALOG_VOLTS

DESCRIPTION:

The SET_ANALOG_VOLTS statement sets the analog engineering units to volts, rather than percent full-scale. All values returned by analog input statements or passed to analog output statements are in units of volts.

The default setting for I/OBASIC is PERCENT. The default setting is restored when a RUN command is issued.



SET_DATE Statement

SYNTAX: SET_DATE(month, day, year)

DESCRIPTION:

The SET_DATE statement will set the system date.

Argument month is a variable or constant of any type that contains the new value for the month of the year. The value of this argument must be supplied by the program. Legal range is 1 to 12.

Argument day is a variable or constant of any type that contains the new value for the day of the month. The value of this argument must be supplied by the program. Legal range is 1 to 31.

Argument year is a variable or constant of any type that contains the new value for the year since 1900. The value of this argument must be supplied by the program. Legal range is 72 to 103, corresponding to the years 1972 to 2003.

NOTE

Although each argument must be within the range specified, the date is not checked for validity. For example, the date could be set to 31-FEB-83 by using the statement SET_DATE(2,31,83). This would be an illegal date.

Statement
SET_THERMOCOUPLE

SET_THERMOCOUPLE Statement

SYNTAX: SET_THERMOCOUPLE(val)

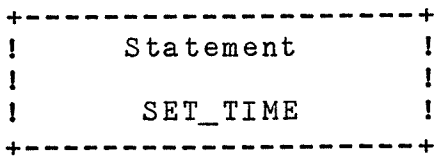
DESCRIPTION:

The SET_THERMOCOUPLE statement sets the thermocouple type and its temperature range to be used with the TEMPERATURE_IN statement. The thermocouple types supported are J, K, and T. Only one thermocouple type may be selected for the TEMPERATURE_IN statement, but it may be changed during program execution. Selecting a narrower temperature range will increase the accuracy of the thermocouple readings.

In the syntax above, argument val is a variable or constant that specifies a code for selecting the thermocouple type and temperature range. It must be in the range from one to six.

val	Type	Range (degrees C)
1	J	-210 to 870
2	J	-210 to 366
3	J	-210 to 185
4	K	0 to 1232
5	K	0 to 484
6	T	-200 to 385

The default value is 1, which selects thermocouple type J and a range of -210 to 870 degrees C. The default value is restored when a RUN command is issued.



SET_TIME Statement

SYNTAX: SET_TIME(val)

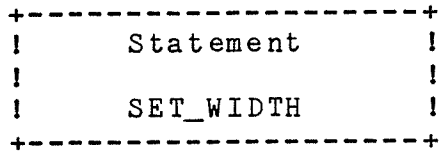
DESCRIPTION:

The SET_TIME statement will set the system time.

Argument val is a variable or constant of any type that contains the new value for the system time, in units of seconds past midnight. The value of this argument must be supplied by the program. Legal range is 0 to 86400.

NOTE

If argument val is of type integer or string and the time is greater than 65,535 seconds past midnight a numeric overflow error will result. To avoid this, specify argument val as a real variable or constant.



SET_WIDTH Statement

SYNTAX: SET_WIDTH(val)

DESCRIPTION:

The SET_WIDTH statement will set the console terminal's right margin. If the number of characters output to the console terminal exceeds the width setting, a return and line feed are automatically generated. The default width setting is 80 characters.

```
+--------------------+
!      Function      !
!                    !
!        SGN         !
+--------------------+
```

SGN Function

SYNTAX: SGN(numeric expression)

DESCRIPTION:

The SGN function is used to determine the sign of an expression. The SGN function returns a +1, -1, or 0 to identify the sign of the specified expression as positive, negative, or zero, respectively.

EXAMPLE:

```
>listnh <ret>
10 a = -8.34
20 b = 45.23
30 c = 0
40 PRINT "A = ";a; "SIGN A = ";SGN(a)
50 PRINT "B = ";b; "SIGN B = ";SGN(b)
60 PRINT "C = ";c; "SIGN C = ";SGN(c)
70 END

>runnh <ret>

A = -8.34 SIGN A = -1
B = 45.23 SIGN B = +1
C = 0     SIGN C = 0

>
```

```
+--------------------+
!      Function      !
!                    !
!        SIN         !
+--------------------+
```

SIN Function

SYNTAX: SIN(numeric expression)

DESCRIPTION:

The SIN function returns the sine of the of the angle specified. The angle is specified in radians.

EXAMPLE:

```
>listnh <ret>
10 REM Convert angle to radians and print sine
20 PRINT "Enter an angle in degrees";
30 INPUT deg_angle
40 REM Convert the angle to radians
50 LET rad_angle = deg_angle * 2 * PI / 360
60 PRINT "The sine of angle A is "; SIN(rad_angle)
70 END

>runnh <ret>

Enter an angle in degrees? 45 <ret>
The sine of angle A is .707107

>
```

Function
SQR

SQR Function

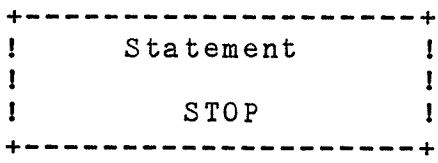
SYNTAX: SQR(numeric expression)

DESCRIPTION:

The SQR (square root) function returns the square root of the specified numeric expression. If the expression specified is negative, an error message is returned.

EXAMPLE:

```
>listnh <ret>
10 READ a, b, c
20 PRINT SQR(a); SQR(b); SQR(c)
30 DATA 16, 2.5, -4.5
40 END

>runnh <ret>

 4         5.65685
?Negative square root at line 20
 0

>
```



STOP Statement

SYNTAX: STOP

DESCRIPTION:

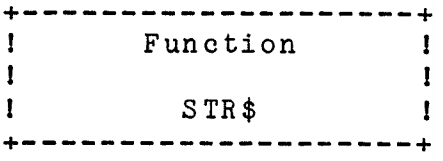
The STOP statement causes program execution to halt and I/OBASIC prints the message:

Stop at line x

Where x is the line number of the STOP statement.

The STOP statement is useful in debugging since variables can be printed, then changed, and program execution resumed with a GOTO statement.

The STOP statement does not close files opened during program execution, therefore, the END statement should be used at the logical (and physical) end of a program.



STR$ Function

SYNTAX: STR$(numeric expression)

DESCRIPTION:

The STR$ function converts the specified numeric expression to its string equivalent. The string value is returned with no leading or trailing spaces.

Command
SUB

SUB Command

SYNTAX: SUB linenumberXstring1Xstring2Xnumber of occurrences

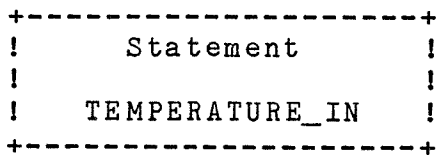
DESCRIPTION:

The SUB command allows you to edit a program line without retyping the entire line.

Linenumber specifies the program line number to be edited. String1 specifies the characters to be replaced by the characters in string2. X is a delimiter which may be any character not found in the original or replacement strings. Number of occurrences is an integer number which specifies the number of times string1 should occur in the line before the substitution is be made.

EXAMPLE:

```
>listnh 10 <ret>
>10 LET z = z + x + y

>sub 10/y/n <ret>                <Substitute n for y

10 LET z = z + x + n             <New Line

>sub 10/z/b/2 <ret>              <Substitute b for 2nd z

10 LET z = b + x + n             <New Line

>
```



TEMPERATURE_IN Statement

SYNTAX: TEMPERATURE_IN(chan, val [,flag])

or

TMPIN(chan, val [,flag])

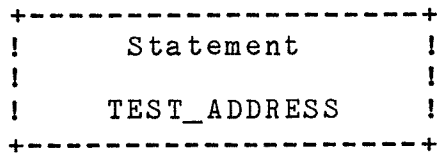
DESCRIPTION:

The TEMPERATURE_IN statement reads one or more channels on the low-level analog input board, returning the values in degrees centigrade.

Argument chan is a variable or constant of any type that contains the analog channel to be read, or the last channel to scan if channel scanning is set. The value of this argument must be supplied by the program. Legal range is 0 to 1023.

Argument val is a variable of any type that will be returned with the temperature value of the analog channel. If argument val is of type real the value is returned in units of degrees centigrade. If argument val is of type integer or string, the value returned is the unconverted binary value of the A/D converter. If argument val is an array then the entire array will be filled with analog channel values. String arrays are not allowed.

Argument flag is an optional integer or real variable. When specified, causes the statement to operate asynchronously in interrupt mode, or DMA mode if an ADAC 1622DMA controller is present. It will be set to a zero prior to the next I/OBASIC program statement, and will be set to a one when all of the analog values have been read. If argument flag is not specified, then the statement will operate synchronously, so that all processing is completed before the next I/OBASIC statement.



TEST_ADDRESS Statement

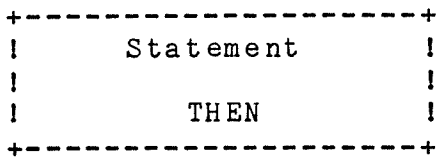
SYNTAX: TEST_ADDRESS(address, val)

DESCRIPTION:

The TEST_ADDRESS statement will test for a valid bus address.

Argument address is a variable or constant of any type that contains the address to be tested. The value of this argument must be supplied by the program, and it must be an even number. Legal range is 0 to 177776 (octal).

Argument val is a variable of any type that will be returned with a one if the address is present on the bus, and a zero if the address is not on the bus.



THEN Statement

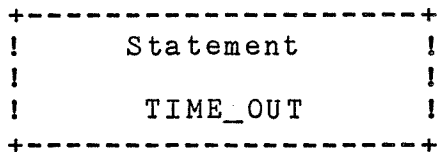
SYNTAX: THEN statement

or

THEN line number

DESCRIPTION:

The THEN statement is used in the expressions IF/THEN or ON/THEN. The THEN statement cannot stand alone. See the IF/THEN and ON/THEN statements.



TIME_OUT Statement

SYNTAX: TIME_OUT(chan, val [,flag])

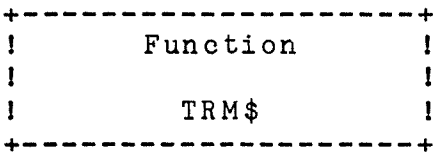
DESCRIPTION:

The TIME_OUT statement starts a software timer.

Argument chan is a variable or constant of any type that specifies the timer channel that will be started. The value for this argument must be supplied by the program. Legal range is 0 to 7.

Argument val is a variable or constant of any type that specifies the time interval, in seconds, for the timer. The value of this argument must be supplied by the program. Legal range is 0 to 540. If argument val is of type real, it may contain a fractional part of a second. The time interval is measured in increments of 1/60th of a second, and is generally accurate to 1/10th of a second.

Argument flag is an optional integer or real variable of any type. When specified, causes the statement to operate asynchronously in interrupt mode, so that an interrupt will be generated after the time interval. Argument flag will be set to a zero prior to the next I/OBASIC program statement, and will be set to a one when the time interval has elapsed. If argument flag is not specified, then the statement will operate synchronously, and behaves the same as the WAIT statement.



TRM$ Function

SYNTAX: TRM$(string expr)

DESCRIPTION:

The TRM$ function is used to remove all trailing blanks from the specified string expression.

EXAMPLE:

```
>listnh <ret>
10 word1$ = "Word 1    "
20 word2$ = "-Word 2"
30 REM Concatenate Word 1 and 2
40 PRINT "Before Trimming: ";word1$ + word2$
50 word1$ = TRM$(word1$)
60 PRINT "After Trimming: ";word1$ + word2$
70 END

>runnh <ret>

Before Trimming: Word 1   -Word 2
After Trimming: Word 1-Word 2

>
```

Command
UNSAVE

UNSAVE Command

SYNTAX: UNSAVE file specification

DESCRIPTION:

The UNSAVE command deletes the specified file and returns the I/OBASIC prompt (>) when complete. Once a file is deleted, it cannot be restored.

Function
VAL

VAL Function

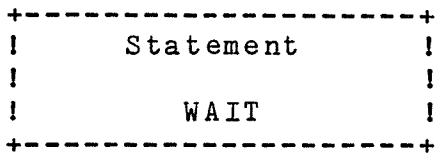
SYNTAX: VAL(numeric constant string)

DESCRIPTION:

The VAL function returns the value of the numeric constant string specified. The string may contain the digits 0 through 9, the letter E for exponential notation, a decimal point, and the + or - symbols.

EXAMPLE:

```
>listnh <ret>
10 number1$ = "-42.123"
20 number2$ = "67.01"
30 PRINT VAL(number1$) + VAL(number2$)

>runnh <ret>

 24.887

>
```



WAIT Statement

SYNTAX: WAIT(val)

DESCRIPTION:

The WAIT statement will delay program execution.

Argument val is a variable or constant of any type that contains the delay time, in units of seconds. The value of this argument must be supplied by the program. Legal range is 0 to 540. If argument val is of type real, it may contain a fractional part of a second. The delay is measured in increments of 1/60th of a second, and is generally accurate to 1/10th of a second.

APPENDIX A

I/OBASIC ERROR MESSAGES

I/OBASIC processes errors in commands and program lines and prints an error message which identifies the error. Error messages are useful in debugging your programs because they help identify the location and nature of the error.

I/OBASIC error messages are identical to standard BASIC error messages. When an error is detected in a command, I/OBASIC returns an error message in the following format:

?Message

When an error is detected in an I/OBASIC program line, I/OBASIC returns an error message in the following format:

?Message at line xx

where xx is the line number of the statement containing the error.

I/OBASIC errors are either fatal or nonfatal. If a fatal error is detected in a program, execution of the program halts, an error message is returned and control returns to the command level. Nonfatal errors will result in an error message being returned, but program execution is not halted.

This appendix lists error messages and their meanings in alphabetical order. The error messages listed here are fatal unless otherwise indicated.

A.1 Command and Program Line Errors

?Argument error

> Arguments in a function or statement do not match those arguments defined for the function or statement in number, range, or type. Ensure that the correct number of arguments are specified, that their values are in the correct range, and that they are the correct type.

?Arrays too large

> Not enough memory is available for the arrays specified in the DIM statements. Reduce the size of the arrays or reduce the size of the program.

?Bad data read

A data item input from a DATA statement or from a file is the wrong data type. Ensure that the DATA statement or the file contains the same data type as specified in the READ or INPUT # statement.

?Bad data - retype from error

Nonfatal error. Item entered in response to an INPUT or INPUT #0 statement is the wrong data type. Retype the item and the program will continue.

?Bad log

Nonfatal error. Expression in LOG or LOG10 function is zero or negative. The function returns a zero and I/OBASIC continues execution of the program.

?Buffer storage overflow

Not enough room available for file buffer in program storage area. Reduce program size.

?Cannot delete file

The file specified in a KILL statement or UNSAVE command cannot be deleted.

?Channel already open

OPEN statement specifies a channel that is already associated with an open file. Ensure that OPEN statements specify correct channel numbers and that files that should be closed are closed.

?Channel I/O error

Accessing data in a file produces an error. Ensure that your peripheral devices and their storage media are working correctly. One possible cause is that the file accessed has zero length.

?Channel not open

A PRINT #, PRINT # USING, INPUT #, IF END #, or CLOSE statement, or a reference to a virtual array file specifies a channel number not associated with an open file. Check that the OPEN statement has been executed and that it specifies the same channel number as the program line with the error.

?Checksum error in compiled program

File produced by the COMPILE command contains a format error. Use a copy of the program created by a SAVE or REPLACE command.

?COMMON out of order

Variables and arrays in a COMMON statement are not listed in the same order as those in a previous program segment. Ensure that all segments have equivalent COMMON statements.

?Control variable out of range

Expression in an ON GOTO or ON GOSUB statement is zero or negative or has a value greater than the number of line numbers listed. Ensure that the expression has a value in the correct range.

?Division by zero

Nonfatal error. An expression includes a division by zero. I/OBASIC substitutes a value of zero for that operation and continues execution of the program.

?END not last

END statement is not the highest numbered program line. This error message is printed when the END statement is executed. Ensure that there is only one END statement in the program and that it has the highest line number.

?Error closing channel

Closing a channel produces an error. Ensure that your peripheral devices and their storage media are working correctly.

?Excess input ignored

Nonfatal error. There are more data items than required by an INPUT or INPUT #0 statement. I/OBASIC ignores the excess items and continues execution of the program. Ensure that data items did not contain an unintended comma (e.g., 5,432 instead of 5.432).

?Exponentiation error

Nonfatal error. An expression includes the operation of raising a negative value to a nonintegral power (e.g., (-45)^.25). This would produce a complex number, which cannot be represented in I/OBASIC. This message is also produced when a negative value is raised to an integral value that has an absolute value greater than 255 (e.g., (-1)^256). In both cases, I/OBASIC substitutes a value of zero for the operation and continues execution.

?Expression too complex

An expression is too complex for I/OBASIC to evaluate in the area it uses for calculations (called the stack). This condition is usually caused by including user-defined functions or nested functions in an expression. The degree of complexity that causes this error varies according to the amount of space available in the stack at the time. Breaking the statement up into several statements containing simpler expressions may eliminate the error.

?File already exists

A file cannot be created with the same name as an existing file. This would result in the deletion of the existing file and this deletion is not allowed.

?File not found

I/OBASIC cannot find the specified file. Ensure that the file specification was typed correctly and that the file exists.

?File privilege violation

This operation includes a restricted file operation.

?Floating overflow

Nonfatal error. The absolute value of the result of a computation is greater than the largest number that can be stored by I/OBASIC (approximately 10E38). I/OBASIC substitutes a value of zero for the operation and continues execution of the program.

?Floating underflow

Nonfatal error. The absolute value of the result of a computation is smaller than the smallest number that I/OBASIC can store (approximately 10E(-38)). I/OBASIC substitutes a value of zero for operation and continues execution of the program.

?FOR without NEXT

The program contains a FOR statement without a corresponding NEXT statement to terminate the loop. Ensure that each loop in the program is terminated with a NEXT statement.

?Illegal channel number

The channel specified is not in the range allowed or the IF END statement specifies a file on a terminal.

?Illegal DEF

There is an error in the DEF statement. Check the format and data types in the argument list and defining expression.

?Illegal DIM

A subscript in a DIM or COMMON statement is not an integer, an array is dimensioned more than once, or an array has more than two dimensions. Ensure that an array specification is in the correct format and appears only once in the COMMON and DIM statements in the program.

?Illegal end of file in compiled program

File produced by the COMPILE contains a format error. Use a copy of the program created by a SAVE or REPLACE command.

?Illegal file length

The file specification is invalid. See your BASYS User's Guide for information on the format of file specifications.

?Illegal I/O direction

Statement attempts to write to an input file or read from an output file. Ensure that the channel number specifies the correct file. If the statement assigns a value to an element of a virtual array file, ensure that the file's OPEN statement does not specify "FOR INPUT."

?Illegal record size

The RECORDSIZE keyword specified in an OPEN statement is invalid.

?Inconsistent number of subscripts

The array is dimensioned with a different number of subscripts than it is referenced by. Ensure that the DIM statement and array references are consistent.

?Input string error

Nonfatal error. A string entered in response to an INPUT statement begins with a quotation mark but is not terminated by the appropriate ending quotation mark. I/OBASIC assigns to the string all the characters between the initial quote and the line terminator and continues execution of the program.

?Integer overflow

An integer variable is assigned a value greater than 32767 or less than -32768 or an integer expression produces a result which exceeds this range. Change the variable or expression to a floating point format.

?Line too long

The line entered is longer than I/OBASIC allows; the line is ignored. If this message occurs when I/OBASIC is reading a program from a file, I/OBASIC stops reading the file. A possible cause is that you entered a line near the maximum size with no spaces; when you save the program, I/OBASIC adds spaces making the line too long. Split the line into several smaller lines.

?Line to long to translate

Lines are translated as they are entered; the line just entered exceeds the area reserved for translating. The line is ignored. If this message is produced while I/OBASIC is reading a program from a file, I/OBASIC stops reading the file. Split the line into several smaller lines.

?Missing subprogram

A nonexistent I/OBASIC statement was specified. Ensure that the statement was typed correctly.

?Negative square root

Nonfatal error. The expression in the SQR (square root) function has a negative value. The function returns a value of zero. I/OBASIC continues execution of the program.

?Nested FOR statements with same control variable

A FOR statement specifies the same control variable as that specified by a FOR/NEXT loop that the FOR statement is nested within. Change one of the variables so that all nested FOR statements have unique variable names. Make sure you change both the FOR statement and the corresponding NEXT statement.

?NEXT without FOR

A NEXT statement is without a corresponding FOR statement. Ensure that each loop starts with a FOR statement and ends with a NEXT statement which specifies the same variable. This error message is also produced if control is transferred into the middle of a loop. FOR/NEXT loops should only be entered by executing the FOR statement.

?Not a valid device

File specification contains an invalid device. See the BASYS User's Guide or Section 5.2 of this manual for further information on file specifications.

?Not enough room

There is not enough room for the file. See the BASYS User's Guide for more information.

?Numbers and strings mixed

String and numeric values appear in the same expression or they are set equal to each other; for example A$=2. Change either the data type of the variable (e.g., A=2) or the expression (e.g., A$= "2") so that they are consistent.

?Out of data

The data list is exhausted and a READ statement requests additional data or the end of a file is reached and the INPUT # statement requests additional data. Ensure that there is sufficient data or test for the end-of-file condition with the IF END statement.

?PRINT USING error

There is an error in the PRINT USING statement caused when the format specification is not a valid string, or is null, or does not contain one valid field. The error is also caused when an attempt is made to print a numeric value in a string field, a string value in a numeric field, or a negative number in a floating asterisk or floating dollar sign field that does not also specify a trailing minus sign. The message is also printed if the items in the list are not separated by commas or semicolon.

?Program too big

The line just entered cause the program to exceed the user area in memory and the line is ignored. Reduce program size. If this error occurs when I/OBASIC is reading a program from a file, I/OBASIC stops reading the file.

?Resequence error

Resequencing the program would cause lines to overlap or existing lines to be deleted, or would create an illegal line number. Re-enter the command with different argument.

?RETURN without GOSUB

A RETURN is encountered before execution of a GOSUB statement. Do not transfer control to a subroutine except by executing a GOSUB or an ON GOSUB statement.

?String storage overflow

Not enough memory is available to store all the strings used in the program. Reduce program size.

?String too long

The maximum length of a string in an I/OBASIC statement is 255 characters. Split the string into several smaller strings.

?Subscript out of bounds

The subscript computed is less than zero or is outside the bounds defined in the DIM statement. Ensure that the expression specifying the subscript is in the correct range.

?Substitute error

There was no separator between the strings in the SUB command. Retype SUB command.

?Syntax error

I/OBASIC has encountered an unrecognizable element. Common examples of syntax errors are misspelled commands, unmatched parentheses, and other typographical errors. This message can also be produced by attempting to read in a program from a file containing illegal characters, in which case I/OBASIC stops reading the file. Retype the program line or ensure that the file contains a valid I/OBASIC program.

?Too many GOSUBs

More than 20 GOSUBS have been executed without a corresponding RETURN statement. Change the program logic so that less GOSUB statements are executed.

?Too many items in COMMON

There are more than 255 variable and array names in COMMON (A, A(100), A%, A%(10,10), A$, and A$(5) are all considered different names). Reduce the number of items in COMMON by converting individual variables to elements of an array or by passing fewer items to the next program segment.

?Undefined functions

A user-defined function has been used and not defined. Define the function. A function is defined only after the RUN command or CHAIN statement is execute.

?Undefined line number

The line number specified in an IF, GOTO, GOSUB, ON GOTO, ON GOSUB, or CHAIN statement does not exist anywhere in the program. Ensure that the line number specified exists in the program.

?Undimensioned array in call

The first reference to an undimensioned array appears as an argument in an I/OBASIC statement. Dimension the array with the DIM statement.

?Use REPLACE

Saving the program would have caused an existing file to be deleted because of identical file specifications. Use a different file specification for the new file or the REPLACE command which will delete the original file.

?Virtual array channel already in use

The DIM # statement specifies a channel number which has already appeared in a DIM # statement. Specify another channel number.

A.2 Function Errors

If you use an I/OBASIC function incorrectly, an error message will be returned. This section describes the conditions which will cause error messages for the various functions.

All functions

The argument used is the wrong type. For example, the argument is numeric and the function expects a string expression. This condition produces "?Argument error".

All functions

The wrong number of arguments were used in a function, or the wrong character is used to separate them. For example, PRINT SIN (X,Y) produces a syntax error because the SIN function has only one argument. This condition produces "?Syntax error".

ASC(string)

String is not a one-character string. This condition produces "?Argument error".

BIN(string)

Characters other than blank, zero, or 1 occur in the string or the value is greater than 2^16. This condition produces "?Argument error".

CHR$(expr)

Expression is not in the range zero to 32767. This condition produces "?Argument error".

EXP(expr)

Value of expression is greater than 87. This condition produces "?Exponentiation error".

FNletter

The function FNletter is not defined (function cannot be defined by an immediate mode statement). This condition produces "?Undefined function".

LOG(expr)

Expression is negative or zero. The function returns a value of zero. This condition produces "?Bad log".

LOG10(expr)

Expression is negative or zero. The function returns a value of zero. This condition produces "?Bad log".

OCT(string)

Characters other than blank or digits zero through 7 appear in the string, or the value is greater than 2^16. These conditions produce "?Argument error".

PI

An argument is included. This condition produces "?Syntax error".

SEG$(string, expr1, expr2)

No additional error conditions.

SQR(expr)

Expression is negative. The function returns a value of zero. This condition produces "?Negative square root".

TAB

Expression is not in the range zero to 32767. This condition produces "?Argument error".

VAL(string)

String is not a numeric constant. This condition produces "?Argument error".

APPENDIX B

I/OBASIC ERROR CODES

This appendix lists the numeric error codes that correspond to possible I/OBASIC program errors. These codes can be obtained using the ERR function after error processing has been enabled by executing the ON ERROR GOTO statement.

ERROR MESSAGE	CODE
Argument error	127
No room for call	126
END not last	125
Arrays too large	124
Undefined line number	123
Syntax error	122
Resequence error	121
Substitute error	120
Illegal DIM	119
Virtual array channel already in use	118
COMMON out of order	117
String too long	116
Too many items in COMMON	115
Function already defined	114
Line too long to translate	113
Program too big	112
Checksum error in compiled program	111
Illegal end of file in compiled file	110
Channel I/O error	109
Buffer storage overflow	108
Channel not open	107
Illegal channel number	106
Channel already open	105
Illegal I/O direction	104
File not found	103
Illegal file specification	102
Not enough room	101
Illegal file length	100
File too short	99
Use replace	98
PRINT USING error	97
Expression too complex	96

Numbers and strings mixed	95
String too long	94
Undefined function	93
Inconsistent number of subscripts	92
Subscript out of bounds	91
String storage overflow	90
Too many GOSUBS	89
Control variable out of range	88
RETURN without GOSUB or RESUME without error	87
Nested FOR statements with same control variable	86
FOR without NEXT	85
NEXT without FOR	84
Illegal in immediate mode	83
Out of data	82
Bad data read	81
Undimensioned array in call	80
Power failure (MICROBASYS only)	79
A/D trigger too fast	78
Hardware not present	77
Event queue overflow	76
Line too long	75
Integer overflow	74
Bad data-retype from error	73
Excess input ignored	72
Input string error	71
Floating overflow	70
Floating underflow	69
Division by zero	68
Negative square root	67
Bad log	66
Exponentiation error	65

APPENDIX C

ASCII CHARACTER EQUIVALENTS

This appendix gives the decimal and octal equivalents of ASCII characters. I/OBASIC users can convert an ASCII value to the corresponding string character with the CHR$ function and can convert a string character to the corresponding ASCII value with the ASC function. ASCII characters are stored internally and in files as eight bits. The eighth bit in normally zero.

DECIMAL VALUE	OCTAL VALUE	ASCII CHARACTER
00	000	NUL (CTRL/@)
01	001	SOH (CTRL/A)
02	002	STX (CTRL/B)
03	003	ETB (CTRL/C)
04	004	EOT (CTRL/D)
05	005	ENQ (CTRL/E)
06	006	ACK (CTRL/F)
07	007	BEL (CTRL/G)
08	010	BS (CTRL/H)
09	011	HT (CTRL/I or TAB)
10	012	LF (NEW LINE or LINE FEED)
11	013	VT (Vertical TAB)
12	014	FF (Form Feed)
13	015	CR (Return)
14	016	SO (CTRL/N)
15	017	SI (CTRL/O)
16	020	DLE (CTRL/P)
17	021	DC1 (CTRL/Q)
18	022	DC2 (CTRL/R)
19	023	DC3 (CTRL/S)
20	024	DC4 (CTRL/T)
21	025	NAK (CTRL/U)
22	026	SYN (CTRL/V)
23	027	ETB (CTRL/W)
24	030	CAN (CTRL/X)
25	031	EM (CTRL/Y)
26	032	SUB (CTRL/Z)
27	033	ESC (ESCAPE)
28	034	FS (CTRL/\)
29	035	GS (CTRL/])
30	036	RS (CTRL/^)

DECIMAL VALUE	OCTAL VALUE	ASCII CHARACTER	DECIMAL VALUE	OCTAL VALUE	ASCII CHARACTER
31	037	US (CTRL/)	80	120	P
32	040	SP (space bar)	81	121	Q
33	041	!	82	122	R
34	042	"	83	123	S
35	043	#	84	124	T
36	044	$	85	125	U
37	045	%	86	126	V
38	046	&	87	127	W
39	047	'	88	130	X
40	050	(	89	131	Y
41	051	)	90	132	Z
42	052	*	91	133	[
43	053	+	92	134	\
44	054	,	93	135	]
45	055	-	94	136	^
46	056	/	95	137	_
47	057	.	96	140	`
48	060	0	97	141	a
49	061	1	98	142	b
50	062	2	99	143	c
51	063	3	100	144	d
52	064	4	101	145	e
53	065	5	102	146	f
54	066	6	103	147	g
55	067	7	104	150	h
56	070	8	105	151	i
57	071	9	106	152	j
58	072	:	107	153	k
59	073	;	108	154	l
60	074	<	109	155	m
61	075	=	110	156	n
62	076	>	111	157	o
63	077	?	112	160	p
64	100	@	113	161	q
65	101	A	114	162	r
66	102	B	115	163	s
67	103	C	116	164	t
68	104	D	117	165	u
69	105	E	118	166	v
70	106	F	119	167	w
71	107	G	120	170	x
72	110	H	121	171	y
73	111	I	122	172	z
74	112	J	123	173	{
75	113	K	124	174	!
76	114	L	125	175	}
77	115	M	126	176	~
78	116	N	127	177	DEL
79	117	O			

APPENDIX D

BASYS FILE EXTENSIONS AND DEVICE NAMES

This appendix lists BASYS file extensions and device names. A DX11 or PX11 file specification consists of a file name and optionally a file extension and device name.

For additional information on file extensions and device names, see the BASYS User's Guide.

File extensions are not limited to those listed below. Those listed below have generally accepted meanings and are useful for identifying the type of information contained in a file. Any combination of one to three letters may be used for a file extension.

FILE TYPE	MEANING
.BAC	I/OBASIC compiled file
.BAK	Editor (or other) backup file
.BAS	I/OBASIC source file
.COM	Indirect command file
.DAT	I/OBASIC data file
.SAV	Executable program file
.SYS	System files and handlers
.TXT	Text file

The two-letter device codes listed below are used to identify possible BASYS System storage devices. These are standard device names used for the Digital Equipment Corporation compatible storage devices comprising the BASYS System.

DEVICE	DEVICE NAME
TU58 Tape Drive	DDn:
RL01/RL02 Hard Disk Drive	DLn:
RX01 Floppy Diskette Drive	DXn:
RX02 Floppy Diskette Drive	DYn:
DX11, PX11 Extended Memory Disks	XMn:

APPENDIX E

I/OBASIC KEYWORDS

Listed below are all of the I/OBASIC keywords. These keywords cannot be used as variable names in an I/OBASIC program. If a "?Syntax error" results when running a newly developed I/OBASIC program, it is possible that a variable name was chosen that is one of the keywords below.

ABORT
ABS
AIN
AINL
ANALOG_IN
ANALOG_LOW_IN
ANALOG_OUT
AND
AOT
APPEND
ASC
ASFILE
ANT
BIC
BIN
BIS
BIT
BIT_CLEAR
BIT_SET
BIT_TEST
BYE
CANCEL_CTLO
CHAIN
CHAR_IN
CHR
CLEAR
CLK
CLOCK_OUT
CLOSE
COMMON
COMPILE
CONVERT_OCTAL
COS
CTLC
NEXT
NOT
OCT
OLD
ON
ON ERROR GOTO
ON EVENT GOSUB
OPEN
OR
OVERLAY
PEEK
PI
POKE
POS
PRINT
RANDOMIZE
READ
REM
RENAME
REPLACE
RESEQ
RESET
RESTORE
RESUME
RETURN
RND
RUN
RUNNH
SAVE
SEG
SET_AINL_GAIN
SET_AINL_NOSCAN
SET_AINL_NOTRIGGER

DAT
DATA
DEF
DEFFN
DEL
DIGITAL_IN
DIGITAL_OUT
DIM
DIN
DISABLE_CTLC
DOT
ENABLE_CTLC
END
ERL
ERR
EVENTRETURN
EXP
FN
FOR
FORINPUT
FOROUTPUT
GET_DATE
GET_TIME
GOSUB
GOTO
IF
INPUT
INT
KILL
LEN
LENGTH
LET
LINE
LINPUT
LIST
LISTNH
LOAD
LOG
LOG10
NAME
NEW
SET_AINL_SCAN
SET_AINL_TRIGGER
SET_AIN_GAIN
SET_AIN_NOSCAN
SET_AIN_NOTRIGGER
SET_AIN_SCAN
SET_AIN_TRIGGER
SET_ANALOG_PERCENT
SET_ANALOG_VOLTS
SET_AOT_NOTOGGLE
SET_AOT_NOTRIGGER
SET_AOT_TOGGLE
SET_AOT_TRIGGER
SET_DATE
SET_THERMOCOUPLE
SET_TIME
SET_WIDTH
SGN
SIN
SQR
STEP
STOP
STORE
STR
SUB
TAB
TEMPERATURE_IN
TEST_ADDRESS
THEN
TIME_OUT
TMPIN
TO
TRM
UNSAVE
USING
VAL
WAIT
XOR