# Apex II ™

By Compsee

## PORTABLE DATA COLLECTION TERMINAL

**Manual #: COAMIIUG0000**

Opertaions & Programming Manual

## PART 3 – Programmer's Manual

COMPSEE


Printed in USA Revision 1.2

***NOTE:***

The software described in this manual is furnished under a license agreement and may be used or copied only in accordance with the terms of that agreement.

Material contained in this manual is for informational purposes only and subject to change without notice. It does not represent a commitment on the part of Compsee

Compsee, Apex, and Apex II are trademark of Compsee. Microsoft is a registered trademark of Microsoft Corporation.

# Table of Contents

# 1 INTRODUCTION

The Apex II Terminals were designed to use BASIC as their programming language. Programming the Apex II Terminals in COMPSEE BASIC can be accomplished by using a computer of any size, equipped with an RS-232-C interface and the appropriate communications software. This versatile Apex II terminal includes the following features:

### Built-in BASIC Interpreter

Complete with your Apex II terminal is the COMPSEE BASIC Interpreter that is built into each device. COMPSEE BASIC is similar to Microsoft's BASIC Interpreter. The difference between the two languages is that the COMPSEE BASIC includes many valuable enhancements to the language to give the programmer complete control over the Apex II terminal. COMPSEE BASIC keeps many of the standard features of BASIC, along with the added features of bar code control and handling, symbology recognition and discrimination, RS-232-C configuration, enhanced file handling, and improved input/output data formatting. Additionally, it gives the programmer the same inter-activity and on-line syntax checking as in all interpreters without the compiling overhead.

### Assembly Language Support

Use of the 80C196NP microprocessor with its 16 MHz clock rate gives COMPSEE BASIC superior speed without compiling. This greatly simplifies the development, debug, and support tasks without compromising program speed and efficiency.

### Error Handling Controls

COMPSEE BASIC includes a number of debugging features which are available in DIAGNOSTIC mode. These features allow the programmer to set break points in the program, change or display the contents of program variables, step through program execution line by line, trace program execution, and branch to a specified line of the program. These features, combined with the interactive nature of an interpreter, reduce the need for program simulators.

### File Processing Capabilities

COMPSEE BASIC supports both random and sequential file access. It can open up to 40 files at one time. Files can contain up to 65535 records with 1 to 2047 bytes per record.

### *Other Features*

Two other features incorporated into the Apex II Terminals are the use of an RS-232-C port and the storage of data in ASCII format. As a result of these standards, your data can be shared with existing application programs, spreadsheet applications, and database programs.

# 2 OVERVIEW OF THIS MANUAL

This part of the manual gives detailed information on the COMPSEE BASIC programming language. Use this manual as a reference while programming to understand a particular statement's effect or syntax. The manual can also be used to explore COMPSEE BASIC by learning about an unfamiliar statement and trying it out in a short program. It is not the intention of this manual to teach BASIC, programming, or use of the PC environment to program. If you need step-by-step instructions for learning to program in BASIC, we suggest you consult the library, bookstore, or software dealer for the materials you may need.

The contents of this manual are described below:

| | |
|---|---|
| Section 1 - 15 | General Information About COMPSEE BASIC, discusses the Compsee's programming environment, COMPSEE keys and their functions, and the Modes of Operation and various keyboard commands. |
| Section 16 | BASIC Instructions, contains reference pages for all BASIC functions and statements. |
| Section 17 - 24 | Reference tables and messages |

Section 17, Error Messages
Section 18, Keypad Table
Section 19, ASCII Table
Section 20, Memory Map
Section 21, LCD Screen Character Set
Section 22, Sample Load Programs
Section 23, Break Modes
Section 24, Communications

# 3 TYPOGRAPHIC CONVENTIONS

The syntax of the statements, functions, system variables, and commands of COMPSEE BASIC interpreter is described in this part of the manual according to the following rules:

1. Keywords are shown in uppercase. However, they may be entered in uppercase or lowercase. If they are entered in lowercase, COMPSEE BASIC converts them to uppercase.

2. Any items shown in lowercase letters must be supplied by the user.

3. Square brackets ([]) indicate optional items.

4. An ellipsis (...) indicates the item may be repeated.

5. All punctuation marks except square brackets are required where shown.

6. All keywords (for instance: OPEN, GOTO) must be followed by a punctuation mark or by a blank. COMPSEE BASIC ignores multiple blanks.

7. Abbreviations used:

   a.e. - arithmetic expression
   f.s. - form string
   c.s. - character string
   c.e. - character expression

8. Functions and system variables are shown on the right side of an assignment in their format descriptions. Functions and system variables can be used anywhere a variable can be used, except on the left side of an assignment.

# 4 OPERATING SYSTEM MODES

To accommodate the various functions and operations that the Apex II is capable of performing, three system modes are provided: IMMEDIATE, RUN and DIAGNOSTIC. Each of these modes represent a different state of functionality. Consequently, it is required that the operator know which commands and which functions can be performed in each of these modes. The following is an extensive explanation of the modes and the operations which can be performed while in those modes.

## *Immediate Mode*

In IMMEDIATE mode, the operating system is looking at the RS-232-C port for direction. This is the mode the Apex II terminal should be in if you plan to send information such as a program, an operating system update, or a file from the host to the Apex II, or from the Apex II to the host.

When an Apex II terminal is in IMMEDIATE mode it is very much like a PC waiting at the C:> prompt— meaning, the application program is not running and the operating system is waiting for instructions. While in this mode, you can use your PC to send commands such as DIR, CLS, and BEEP to the Apex II.

When COMPSEE BASIC is in IMMEDIATE mode, a statement can be entered and executed immediately, or it can be entered and stored away for later execution in RUN mode. The statement must be terminated by a carriage return (ASCII code 13) and a linefeed (ASCII code 10). If the statement is to be stored for later execution, it must be preceded by a line number. If no line number precedes, the statement executes immediately.

For example, the following statement executes immediately, since the statement "PRINT 5" is not preceded by a line number:

```
PRINT 5
5
>
```

In this case, "PRINT 5" is preceded by a line number. Before this statement executes, COMPSEE BASIC must be placed in RUN mode (using the RUN command). When the Apex II terminal is in RUN mode, the application program stored in memory is executed.

```
10    PRINT 5
>RUN
5
>
```

It is necessary to be in IMMEDIATE mode to :

- receive a BASIC program, or
- change/edit a BASIC program in Apex II memory via a PC, or
- receive a new operating system, or
- interact with the operating system (via RS-232-C)

Terminals in the IMMEDIATE mode are those which:

- are shipped directly from COMPSEE
- are re-booted.
- the word QUIT (diagnostic command) has been typed on an Apex II that was in DIAGNOSTIC mode. This causes the scanner to drop into this mode.

To determine if the terminal is in this mode:

- Press <ALT> <V> . This should produce a message on the Apex II's screen containing the operating system version, or
- press <ALT> <X>. This should produce the MODE setting control screen.

Either one of the above will indicate that a terminal is in the IMMEDIATE mode. If the scanner doesn't produce one of these messages, then you may be in the RUN or DIAGNOSTIC modes.

### ***Run Mode***

The RUN mode is where the Apex II terminal is being controlled by the instructions of a BASIC program. The operating system is ultimately controlling the terminal; however, all direction is coming from the user's program. While in this mode, keyboard inputs, RS-232-C communications, file manipulation, and other functions are the responsibility of the BASIC program running within the terminal. All input to the terminal (keyboard/scanner/ RS-232-C) are accepted, processed, or ignored by the BASIC program. For example, if an inventory program is running on the Apex II terminal, and the PC sent the DIR command through the RS-232-C port to the Apex II, the operating system will not receive it since it is now considered data by the BASIC program.

You must be in RUN mode to:

- execute the instructions of a BASIC program.

You can get into RUN mode by:

- sending the RUN command to the Apex II via the RS-232-C port, or
- press <ALT> <R> or
- typing <G> and the program line number will return you to the RUN mode.

You can determine if the terminal is in this mode by :

- Pressing <ALT> <C> will produce a PRESS ENTER message on the Apex II. This key sequence is used to access the DIAGNOSTIC mode and would only produce this message in the RUN mode.

### ***Diagnostic Mode***

In DIAGNOSTIC mode, the operating system is looking at the Apex II's keyboard for direction. This mode is used for debugging a BASIC application program, single stepping through a program, input values for variables, printing the values of variables, setting break points, etc.

A programmer would use this mode to aid in debugging a BASIC program running within the Apex II. Once a unit is placed in DIAGNOSTIC mode, the operating system is once again in control of the Apex II terminal. Diagnostic mode supports a number of options that allow the user to directly manipulate limited functions of the operating system. This includes checking the contents of variables, changing the value of variables, changing the flow of the program, and tracing the flow of the program.

You must be in Diagnostic mode to:

- use diagnostic functions (as defined in this COMPSEE BASIC Interpreter Programmer's Manual), or
- to break into the IMMEDIATE mode from the keypad.

You can get into Diagnostic mode by:

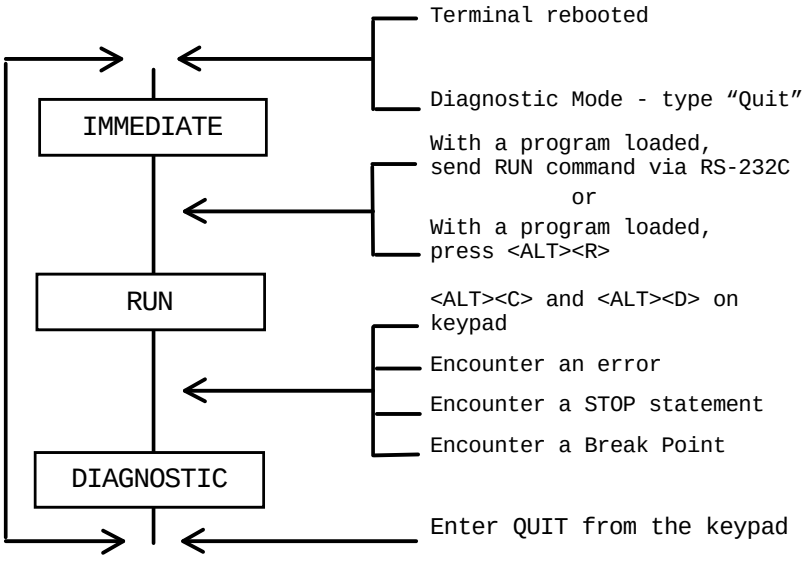
- pressing <ALT> <C> followed by <ALT> <D> from the keypad, or
- having a STOP statement in the BASIC program, or
- setting a BREAK point (diagnostic mode) in the program, or
- running a BASIC program within the terminal that encounters an un-trapped error. This will cause the scanner to break into this mode.

You can determine if the terminal is in this mode by:

- viewing the Apex II's display. You should find the Apex II prompt (>) on the second line. Additionally, when you press keys on the Apex II keypad, the corresponding character will be displayed on the Apex II's screen.

***The following diagram represents mode flow:***

Each of these modes represent a different state of functionality.




# 5 KEYBOARD COMMANDS

While in IMMEDIATE mode, COMPSEE BASIC recognizes certain ASCII characters received from the keyboard as special keyboard commands. When an Apex II terminal is first powered up, the default keys are assigned the proper ASCII values to invoke the keyboard commands. The user is free to reassign these ASCII values to other keys. See Section 18 - Keypad Lookup Table for further information on reassigning key values.

The following is a quick reference guide for COMPSEE keyboard commands:

| FUNCTION | KEYBOARD COMMAND |
|---|---|
| Reboot Apex II - Program remains in FLASH | Hold down ALT•(period); Then turn Apex II ON |
| Reboot Apex II - Program is erased from FLASH | Hold down ALT SPC; Then turn Apex II ON |
| Check Version | ALT V |
| Display communication parameters | ALT X |
| Set communication parameters when they are displayed | X to change first number<br>Y to change Second number<br>Press <ENTER> to exit |
| Clear Buffers | ALT Z |
| Run the Program | ALT R |
| Stop a Program that is Running<br>(unit is then in debug mode) | ALT C (followed by) ALT D |
| Contrast Level Up | ALT I |
| Contrast Level Down | ALT J |
| Save program to FLASH | ALT S |

### ***Keyboard Command: CONTRAST LEVEL UP/DOWN***

ASCII value: 11 (level up)
7 (level down)

Default key: <ALT> <I> (level up)
<ALT> <J> (level down)

Definition: Keystrokes assigned by the operating system to control contrast. These keystrokes increase/decrease the contrast. The contrast controls operate similar to the ALT function. See the Diagnostic topic of this section for information on the <ALT> <C> function. See the <ALT> <H> function and SYSPARMS Statement for additional rules required to operate the <ALT> <I> and<ALT> <J> functions.

### ***Keyboard Command: RUN***

ASCII value: 18

Default key: <ALT> <R>

Definition: Execute the application program stored in memory. (See the RUN command for further details.)

### ***Keyboard Command: VERSION***

ASCII value: 22

Default key: <ALT> <V>

Definition: Display the COMPSEE BASIC version level of the operating system stored in RAM for three seconds. The content of the screen is saved before the version is displayed. Three seconds later, the content of the screen is restored. (See the VER$ system variable for further details.)

### ***Keyboard Command: MODE***

ASCII value: 24

Default key:<ALT> <X>

Definition: Change the communications parameters of the RS-232-C port. The content of the screen is saved and the following message is displayed on the first line:

MODE X,Y

where X denotes the baud rate, and Y is the status. The data mode is always set to binary. (See the MODE command for further details.)

Change the communications parameters on the Apex II's RS 232 port as follows:

1. Press <ALT><X> to view the current parameters. A message similar to the one below will appear: A table of mode settings and their characteristics is provided at the end of this MODE Command topic.

   MODE 5,3

2. Ensure that the cursor is flat (the cursor is shown next to the second number), which means it must appear as an underscore, not a rectangle. Press <ALT> until the flat cursor appears.

3. Change the first number by pressing <X> until the correct baud rate value appears on the screen.

   The following is a chart of all the possible "X" values and the baud rate that they represent:

| "X" Value | Baud Rate |
|---|---|
| 0 | 300 baud |
| 1 | 600 baud |
| 2 | 1,200 baud |
| 3 | 2,400 baud |
| 4 | 4,800 baud |
| 5 | 9,600 baud |
| 6 | 19,200 baud |

4. Change the second number by pressing <Y> until the desired number is reached.
   The following is a chart of all the possible "Y" values and what they represent:

| Status Byte "Y" Value | Flow Control | Data Bits | Parity | Stop Bits |
|---|---|---|---|---|
| 0 | RTS/CTS | 8 | N | 1 |
| 1 | XON/XOFF | 8 | N | 1 |
| 2 | RTS/CTS | 7 | E | 1 |
| 3 | XON/XOFF | 7 | E | 1 |
| 4 | RTS | 8 | O | 1 |
| 5 | XON/XOFF | 8 | O | 1 |
| 6 | RTS/CTS | 7 | O | 1 |
| 7 | XON/XOFF | 7 | O | 1 |
| 8 | RTS/CTS | 8 | N | 2 |
| 9 | XON/XOFF | 8 | N | 2 |
| 10 | RTS/CTS | 7 | E | 2 |
| 11 | XON/XOFF | 7 | E | 2 |
| 12 | RTS/CTS | 8 | O | 2 |
| 13 | XON/XOFF | 8 | O | 2 |
| 14 | RTS/CTS | 7 | O | 2 |
| 15 | XON/XOFF | 7 | O | 2 |

5. Press <Enter> to save the new settings.

NOTES:

1. When modifying the communications parameters, DO NOT power down the Apex II terminal until you have completed all the steps listed above. If you power down prior to completing the procedure and then power the unit on again, the last command entered will still be displayed.
2. In some cases, after you have changed the communication parameters, the screen buffer may contain the previous communications parameters. This may lead you to believe the terminal did not recognize the new settings. To verify that the new settings are stored, press <ALT><X>. The new settings will appear. Press <Enter> to return to the previous screen.
3. While the keyboard is executing the MODE command, <ALT><X> steps through the baud rate values and <ALT><Y> (ASCII value 25) steps through the status values. The <Enter> key (ASCII value 13) terminates the MODE command, sets the RS-232-C port, and restores the screen. The keyboard remains in ALT mode until the <ALT> key is pressed.

### ***Keyboard Command: CLEAR BUFFERS***

ASCII value: 26

Default key: <ALT> <Z>

Definition: Clear the four COMPSEE BASIC buffers:
1. RS-232-C receive
2. RS-232-C transmit
3. Keyboard
4. Bar Code

***NOTE***

If you attempt to scan a bar code when the buffer is full, the terminal will give ten short beeps. Press <ALT> <Z> to temporarily clear the buffers.

# 6 DIAGNOSTIC MODE

COMPSEE BASIC enters DIAGNOSTIC mode if an application program is running, and one of the following is encountered:

1. STOP statement
2. An error
3. A <ALT><C>, <ALT><D> sequence issued from the Apex II terminal keypad
4. A break point

See BREAK MODES, Section 22 for further information on the modes COMPSEE BASIC can enter after a break.

Single-stepping a program also causes COMPSEE BASIC to enter the DIAGNOSTIC mode at the start of every line.

***NOTE***

The default keypad sequence for <Control> <C>, <Control> <D> is <ALT> <C>, <ALT> <D>. Since the keypad is programmable, these sequences may not always apply. For this reason, hereafter the manual will refer to these sequences by their values only. The <ALT> <C>, <ALT> <D> sequence does not work if you are in an error routine.

When <ALT> <C> is pressed, COMPSEE BASIC saves the contents of the screen then, on the first line, displays:

Press <ENTER>

When the DIAGNOSTIC mode is entered, the screen contents are saved and the first line displays:

BREAK (line number)

Line number is the last line COMPSEE BASIC processed (for a STOP, error, and <ALT><C>, <ALT><D>key sequence) or the line COMPSEE BASIC is about to process (for a break point or single step). The prompt character > is displayed in the first column of the second line indicating COMPSEE BASIC is ready to accept DIAGNOSTIC commands. All commands are entered through the Apex II terminal keypad and terminated by pressing the <ENTER> key.

All DIAGNOSTIC mode commands are reset by the RUN command.

An application program can be debugged using COMPSEE BASIC statements in IMMEDIATE mode. The following DIAGNOSTIC commands have a direct relation to a COMPSEE BASIC statement:

| DIAGNOSTIC COMMAND | COMPSEE BASIC STATEMENT |
|---|---|
| G | Go to Line Number |
| I | Let (define a variable) |
| B | Break (1num, 1num) |
| T | Trace on/off |
| P | Print (unformated) |
| QUIT | Quite to Immediate Mode |
| S | Single Step |

Break points cannot be set in IMMEDIATE mode, nor can the program be single stepped. Trace also cannot be set in IMMEDIATE mode.

The syntax of the DIAGNOSTIC commands is described according to the following rules:

1. Any items shown in lower case letters must be supplied by the user.

2. Items in square brackets ([]) are optional.

3. An ellipsis (...) indicates the item may be repeated.

4. All punctuation marks except square brackets are required where shown.

5. Blanks are ignored.

1. All commands, with the exception of QUIT, are one letter.

### ***DIAGNOSTIC Command - Break***

DEFINITION: Set or clear break points.

FORMAT: B [Lnum,Lnum,...]

Lnum - An application program line number.

NOTES:

1. COMPSEE BASIC stops the execution of the application program before the break point is executed and enters DIAGNOSTIC mode.
2. Eight separate break points can be set at one time; B 10,20,10 counts as two break points.
3. The letter B without line numbers clears all break points.
4. Break points are set until explicitly cleared by a B or implicitly cleared by a RUN command.
5. Certain break points do not stop the application program. If a break point is set at the line after an error and a G command is executed, the program does not break.

Also, if a break point is set at the line from which execution is continued, the program does not break. For example, if a break point is set on line 20 (B 20) and execution is continued on line 20 (G 20), the program does not break. See the Go command for further details on program continuation.

EXAMPLE:

```
B 10,20,35
```
Sets a break point at line 10, line 20, and line 35.

### ***DIAGNOSTIC Command - Go***

DEFINITION: Continue the application program.

FORMAT: G [Lnum]

Lnum - The application program line to continue from.

NOTES:

1. A G without a line number continues the program from where execution was halted.
2. Execution can be continued from any line; however, care must be taken so that the program is not continued from an illogical point (e.g., the middle of a subroutine without first executing a GOSUB, or a DIM statement).

EXAMPLES:

Example 1 - Continues execution from where the program was halted.

```
G
```

Example 2 - Continues execution from line 20.

```
G 20
```

### ***DIAGNOSTIC Command - Input***

DEFINITION: Assign a value to a variable.

FORMAT: I variable = value

variable - A scalar or array element.

value - An arithmetic expression or a character expression. The type of expression must match the type of variable.

NOTES:

1. See the LET statement for more information on assignments.

EXAMPLE:

```
I X = 9
```

Assigns a 9 to variable X.

### ***DIAGNOSTIC Command - Print***

DEFINITION: Print a value.

FORMAT: P value

value - The character expression, arithmetic expression, or variable to be printed.

NOTES:

1. Action suspends after the print until a key is pressed so that long string variables do not completely scroll off the screen before they can be viewed.

EXAMPLES:

Example 1 - Prints the variable A$.
```
P A$
```

Example 2 - Prints the length of the string in variable A$.
```
P LEN(A$)
```

### ***DIAGNOSTIC Command - QUIT***

DEFINITION: Exit DIAGNOSTIC mode and enter IMMEDIATE mode.

FORMAT: QUIT

NOTES:

EXAMPLE:

```
QUIT
```

Exits DIAGNOSTIC mode.

### ***DIAGNOSTIC Command - Single Step***

DEFINITION: Executes a single program line.

FORMAT: S

NOTES:

1. An S command causes COMPSEE BASIC to execute the next application program line then return to DIAGNOSTIC mode.

EXAMPLE:

```
S
```
Executes the next program line then returns to DIAGNOSTIC mode.

### *DIAGNOSTIC Command - Trace*

DEFINITION: Send the line number of the line COMPSEE BASIC is processing out the RS-232-C port.

FORMAT: T

NOTES:

1. Trace toggles on and off. The first T command invokes trace; the next T command disables trace.

EXAMPLE:

```
T
```
Toggles the status of race.

# 7 CHANGES TO THE APPLICATION PROGRAM

An application program can be altered once it is stored in memory. Program lines can be edited in IMMEDIATE mode via the RS-232-C port. A program line cannot be greater than 255 characters; also, the COMPSEE BASIC tokenized line cannot be greater than 255 bytes. COMPSEE BASIC converts all lowercase letters to uppercase, except letters within quotes or remarks.

### *Adding a Line*

Enter a valid line number (between 1 and 65535, inclusive), followed by at least one non-blank character, followed by a carriage return/linefeed (CRLF). A line can contain more than one statement if the statements are separated by a colon. For example, if the line below is entered, it is saved in memory as line 10 of the application program:

```
10    A = : GOTO 10
```

The line above is not a valid COMPSEE BASIC statement; an arithmetic expression must follow the equals sign. But, no error is given when the line is entered because COMPSEE BASIC differentiates between tokenized errors and execution errors.

Errors that prevent COMPSEE BASIC from converting the ASCII line to COMPSEE BASIC tokens are flagged as tokenized errors. Tokenized errors are sent out the RS-232-C port:

```
10    A$ = "hello
      A$ = "hello
         ---------X
      *     ERROR 9
```

Error numbers 1 through 12 are tokenized errors. But not all tokenized errors give an echo of the line as in the above example. Error 11 (out of memory) gives no line echo because the line which causes the Apex II terminal to run out of memory does not itself contain an error. But, when COMPSEE BASIC tried to store the line, an error resulted. Error 1 (line too long) also gives no echo.

Execution errors are not flagged until the line is actually executed. Execution errors appear on the screen. In IMMEDIATE mode, the error appears immediately because the line executes immediately.

If the line is stored in memory, the error does not appear until the line is executed via a RUN command. An execution error in RUN mode also prints the error line number on the screen. All errors above error 12 are execution errors. Error 2 (Out of Table Memory) and error 11 (Out of memory) can also occur during execution.

### *Replacing a Line*

If the line number at the beginning of the line already exists in memory, COMPSEE BASIC replaces the existing line with the new line. The rules for errors in the replace line are the same as adding a line.

### *Deleting a Line*

A line number with no text following or a line number with only blanks following is a delete line. COMPSEE BASIC finds the line number in the application program and deletes it. If the line number is not in the program, an undefined line number error (Error 12) results. A series of lines can be deleted using the DELETE command. The entire application program can be deleted using a NEW command.

If the application program is very large and extensive editing is done, some parts of memory may become unusable due to COMPSEE BASIC's memory management. Also, the delete of a large program is a slow process; the Apex II terminal may power down during the delete (turning the unit back on will continue the delete). The recommended approach in this case is to issue a NEW command to delete the old program and reload the application program with the changes. See MEMORY MAP, Section 20 for a further discussion of COMPSEE BASIC's memory management.

# 8 CHARACTER SET

The COMPSEE BASIC character set consists of alphabetic, numeric, and special characters. These special characters have specific meaning to COMPSEE BASIC.

| | | | |
|---|---|---|---|
| | Blank | * | Multiplication |
| = | Equal to or assignment | > | Greater than |
| < | Less than | / | Division |
| + | Addition or concatenation | - | Subtraction or negation |
| ( | Start of array or substring | ) | End of array or substring |
| : | Statement separator, substring indicator, or label terminator | , | Comma |
| ; | Suppress carriage return/linefeed | # | File number designator |
| $ | String variable designator | . | Decimal point |
| " | String delimiter | ' | Remark delimiter |

If a special character appears within quotes, COMPSEE BASIC treats it as part of the string.

# 9 NAMES

COMPSEE BASIC recognizes four types of names: variables, labels, keywords, and files.

File names are handled differently than the other three because file names are within quotes (see the OPEN statement in this manual for further details). File names can be from one to eight ASCII characters with codes between 33 and 126 inclusive.

Variables, labels and keywords must be delimited by blanks or special characters so COMPSEE BASIC can interpret them correctly.

Variables and labels have the same restrictions. They must consist of alphanumeric characters (A-Z and 0-9). A label cannot be greater than eight characters. A string variable can have eight characters plus the string designator character: $.

Variables and labels cannot start with a number. A label must be the first word on the program line and be immediately followed by a colon or follow a label keyword (see GOTO, GOSUB, RESUME, and RESTORE). If the label does not follow these rules, COMPSEE BASIC interprets the label as a variable.

# 10 LABELS

Labels speed program execution time. The address of the label in memory is stored in a table. When a label is used for a program branch (e.g., a GOTO or GOSUB), the label is found in the table and its associated address in memory is retrieved. COMPSEE BASIC immediately branches to the label and the associated program line. If a line number is used, COMPSEE BASIC must do a line by line search from the beginning of the program until the line number is found.

# 11 NUMBERS

COMPSEE BASIC has eight digit significance for numbers. Any numbers which cannot be expressed in eight digits are rounded off. Commas are not allowed within a number. A valid number is:

1. 1 e-127 < = number < = 9.9999999 e + 126 for numbers.
2. -9.9999999 e + 126 < = number < = 1e-127 for negative numbers
3. 0

The sign of a positive number is shown as a blank; a plus sign ('+ ) causes an error.

A = 14.89 valid
A = +14.89 invalid

The sign of a positive exponent is a plus sign. The sign of a negative number of exponent is a minus sign. An exponent without a sign is assumed to be positive.

The limited precision of COMPSEE BASIC may cause incorrect results when performing operation on numbers greater than 8 digits.

EXAMPLES:

Example 1

```
A = 99999999 -  99999998        A equals 1
A = 100000000 - 99999998        A equals 0
```

Example 2

```
B = 3.0 E-127 -   2.0 E-127     B equals 1.0 E-127
B = 3.0 E-127 - 2.1 E-127       B equals 9.0 E-0
```

Example 3

```
C = 3.0 E-127 *  3`             C equals 9.0 E-127
C = 3.0 E-127 * .3              C equals 9.0 E-0
```

Example 4

```
D = 3.0 E126 + 3.0 E126          D equals 6.0 E126
D = 3.0 E126 * 2                 Causes an ERROR 28
```

If a statement requires an integer as an input parameter and the number used is a real number, COMPSEE BASIC truncates the number. Consider the statement:

LOCATE 1.9,1.4

In this case, COMPSEE BASIC truncates the numbers to interpret the statement as:

LOCATE 1,1

# 12 STRINGS

String variables and constants can range from 0 to 255 characters, inclusive. The default maximum length of a string variable is 18 characters. If the variable is to hold a larger string, the variable must be dimensioned in a DIM statement.

Strings can be concatenated to form a larger string. The resulting string cannot be greater than 255 characters. Concatenation is done with a plus sign. Consider this example:

```
10 A$ = "Hello " : B$ = "there"
20 C$ = A$ + B$
```

In this case, C$ contains "Hello there".

Substrings are handled differently in COMPSEE BASIC Interpreter than in most other BASICs. A substring is represented in the form:

```
T$(X:Y)
```

where X is the start position and Y is the end position of the substring within string T$. A substring can be referenced on either side of an equation. Consider this example:

```
10 A$ = "Hello there"
20 B$ = A$(7:11)
30 A$(2:5) = "ELLO"
```

Here, B$ contains "there" and A$ contains "HELLO there".

Substrings have five referencing rules:

1. A start position of 0 is considered position 1. A negative start or end position causes an error. Both positions are truncated to integers.
2. If the start position is greater than the string length, the substring addressed follows the last character of the string. If T$ equals "1234" then T$(5:7) = "567" results in T$ = "1234567".
3. An end position greater than the string length is considered to be equal to the string length.
4. If the start position is greater than the end position and the substring is on the left side of an expression, the expression being assigned is inserted into the string immediately before the specified start position of the substring. For example, if T$ equals "1234", then T$(3:2) = "ABC" results in T$ = "12ABC34". If the start position is greater than the end position, a null string is returned. For example, if T$ equals "1234", then A$ = T$(3:2) = "ABC" results in a null string assigned to A$.
5. The location of a substring assignment must be allocated. For example, if T$ is dimensioned to 18 characters and equals "1234" then T$(9:10) = "AB" results in T$ = "1234AB", not
   T$ = "1234`  AB".

# 13 ARRAYS

Arrays must be explicitly dimensioned in a DIM statement. One dimensional and two dimensional arrays are allowed. COMPSEE BASIC uses option base 0 when dimensioning arrays; i.e., the first element in an array is element 0. The following statement dimensions an array that has 11 elements; 0 through 10:

```
DIM A(10)
```

The following array is dimensioned to have 12 elements: 0,0 0,1 0,2 0,3 1,0 1,1 1,2 1,3 2,0 2,1 2,2 2,3:

```
DIM A$(2,3)
```

Note that arrays are stored in the row, column format. In the above example, A$(1,3) is the eighth element.

# 14 OPERATORS

COMPSEE BASIC operators are divided into four categories: arithmetic, relational, logical, and functions. The COMPSEE BASIC functions are described later in this manual.

## *Arithmetic*

The arithmetic operators in order of precedence are:

*, / Multiplication and division

+, - Addition (or string concatenation) and subtraction

## *Relational*

The relational operators are:

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

Relational operators are all on the same level of precedence. Strings are compared by using the ASCII codes of their characters.

COMPSEE BASIC matches the ASCII codes of the strings character by character from the beginning to the end of each string. As soon as a different ASCII character is found in the same position as both strings, the string with the lower ASCII value is said to be less than the string with the higher ASCII value (for example, PRICE is less than PRIZE). If two strings are equal up to a given point and one is shorter in length than the other, the shorter string is said to be less than the longer string (for example, PRICE is less than PRICES). Two strings of equal length and matching ASCII codes are said to be equal.

### ***Logical***

The logical operators are:

| | |
|---|---|
| AND | Bit-wise Boolean AND |
| OR | Bit-wise Boolean OR |
| XOR | Bit-wise Boolean exclusive OR |

Logical operators are all on the same level of precedence. The operators all have two operands; e.g., ABEL AND CAIN. The operands must be in the range of 0 to 65535 or an invalid function parameter error results. COMPSEE BASIC converts the operand to an integer, truncating when necessary, before doing the bit-wise operation. COMPSEE BASIC uses a zero to indicate a false operation and a non-zero to indicate a true operation. System functions that return a true or false result (e.g., LOC) use 0 as false and -1 as true.

However, relational operators return a 0 as false (for example, PRINT 1 = 0) and 65535 as true (for example, PRINT 1 = 1).

COMPSEE BASIC's overall order of precedence, from highest to lowest, is:

Unary negation (NOT)
Multiplication and division
Addition and subtraction
Relational operators
Logical operators
Functions

# 15 GENERAL RESTRICTIONS AND INFORMATION

COMPSEE BASIC puts certain limitations on the programming of the Apex II terminal:

| | Vex | ss2.0xb | 1.xva |
|---|---|---|---|
| Maximum number of variable names: | 800 | 500 | |
| Maximum number of labels: | | 400 | 200 |
| Maximum number of nested loops: | | | 11 |
| Maximum number of nested subroutines: | | | 13 |
| Limit on IF-THEN-ELSE/GOSUB nests: | | | 6 |
| Maximum number of files in the directory: | | | 40 |
| Maximum Label Length/Variable Length: | | | 8 |

A nested loop is the execution of a FOR/NEXT or WHILE/WEND loop from within another FOR/NEXT or WHILE/WEND loop. The following program has loop nests of three:

```
10    FOR I = 1 TO 5
20    FOR J = 1 TO 3
30    FOR K = 1 TO 2
40    A = A + (I + J + K)
50    NEXT K
60    NEXT J
70    NEXT I
```

A nested subroutine is the execution of a subroutine from within another subroutine. The following program has a subroutine nest of two:

```
10    GOSUB TEST
20    STOP
50    TEST:  GOSUB TEST1
60    RETURN
70    TEST1: PRINT "HELLO"
80    RETURN
```

An IF-THEN-ELSE/GOSUB nest occurs when a GOSUB statement is executed from within an IF-THEN-ELSE statement. The following program has an IF-THEN-ELSE nest of two:

```
10    GOSUB TEST
20    STOP
100   TEST: IF A=2 THEN GOSUB TEST1:
      PRINT C ELSE PRINT D:RETURN
200   TEST1: PRINT E:RETURN
```

An area in memory (a total of 16 bytes) has been set aside for use by the user:

| | |
|---|---|
| Start address: | 22282 |
| End address: | 22297 |

# 16 BASIC INSTRUCTIONS

## *ABS Function*

DEFINITION: Return the absolute value of an arithmetic expression.

FORMAT: x = ABS(num)

num - An arithmetic expression with a valid value described in the number section of this manual.

NOTES:

EXAMPLE:

```
10 A = 25
20 B = -233
30 PRINT A,B,ABS(A*B)
RUN
```

Sends 25, -233, and 5825 to the RS-232-C port.

## *AIM*

DEFINITION: Set the duration of the aiming dot for a long range laser.

FORMAT: AIM x

x - An arithmetic expression which represents the aiming dot duration in increments of 10 ms. Valid values are 0 to 255.

NOTES:

1. The default duration is 1 second (AIM 100).
2. A duration of 0 (zero) will disable the aiming dot.
3. If an AIM command is sent to a standard range laser, the Apex II will return an ERROR 87 message.

EXAMPLES:

Example 1

```
AIM 200
```

Sets the aiming dot duration to 2 seconds (200 times 10 ms).

Example 2

```
AIM 50
```

Sets the aiming dot duration to one-half second (50 times 10 ms).

Example 3

```
AIM 0
```

Disables the aiming dot.

### ***ASC Function***

DEFINITION: Return the ASCII code, in decimal, for the first character of a string.

FORMAT: x = ASC(string)

string - A character expression

NOTES:

1. ASC returns a number between 0 and 255.
2. A null string returns a 0.
3. The string may be any length, up to 255 characters, but only the value for the first character is returned.
4. The ASC function is the inverse of the CHR$ function.

EXAMPLE:

```
10 A = ASC("A")
20 PRINT A
RUN
```

Sends a 65 to the RS-232-C port.

### ***ATN Function***

| This function is not supported. |
|---|

## ***BARCODE$ System Variable***

DEFINITION: Return the contents of the bar code buffer.

FORMAT: x$ = BARCODE$

NOTES:

1. BARCODE$ returns the first bar code in the buffer; the bar code buffer can contain more than one bar code at a time.
2. If the bar code buffer is empty, a null string is returned.
3. BARCODE$ removes the bar code from the buffer. Therefore, the bar code should be assigned to a variable if the bar code is to be used later (see Example 2).
4. Every bar code has a one character prefix which describes the bar code type:

| | | | |
|---|---|---|---|
| A - | Code 3 of 9 | H - | UPC-E |
| B - | Interleaved 2 of 5 | I - | EAN-8 |
| C - | Codabar | J - | EAN-13 |
| D - | Code 128 | K - | MSI Plessey |
| E - | Code 93 | L - | External Bar Code |
| F - | Code 11 | Z - | Unable to decode the bar code |
| G - | UPC-A | | |

5. A RUN command clears the barcode buffer.

EXAMPLES:

Example 1 - Sends the first bar code in the bar code buffer out the RS-232-C port. The bar code is prefixed with a one character bar code type.

```
PRINT BARCODE$
```

Example 2 - Ignores garbage in the bar code buffer. If the bar code buffer is empty (A$="") or the last scan did not decode (A$="Z"), the program loops back to fetch another bar code. The program loops until the bar code buffer contains a valid bar code.

```
10 A$ = BARCODE$
20 IF A$ = ""OR A$ = "Z" THEN GOTO 10
30 PRINT A$
RUN
```

## ***BEEP***

DEFINITION: Sound a ½ second tone.

FORMAT: BEEP

NOTES:

1. If a BEEP is followed by a SIGNAL 2, SIGNAL 3, or SIGNAL 4 statement, the BEEP tone is lost; only the SIGNAL tone is heard.
2. BEEP statements are additive (see Example 2).

EXAMPLES:

Example 1 - Sounds the tone when the loop variable I is 10.

```
10 FOR I = 1 TO 10
20 IF I = 10 THEN BEEP
30 NEXT I
RUN
```

Example 2 - Sounds the tone for approximately 1.5 seconds.

```
BEEP : BEEP : BEEP
```

***CALL***

| This function is not supported. |
|---|

***CHR$ Function***

DEFINITION: Convert an ASCII code to its character equivalent.

FORMAT: x$ = CHR$(n)

n - An arithmetic expression with a value between 0 and 255, inclusive.

NOTES:

1. CHR$ returns a single character string.
2. CHR$ is the inverse of the ASC function.
3. For PRINT #0, COMPSEE BASIC uses the LCD character table located in Section 20.

EXAMPLES:

Example 1 - Appends a carriage return and linefeed to the string "PAGE 1".

```
10 A$ = CHR$(13)
20 B$ = CHR$(10)
30 C$ = "PAGE 1" + A$ + B$
RUN
```

Example 2 - Allows input of a number from 1 to 26 and prints the corresponding letter of the alphabet.

```
10 INPUT$ #1,2,T$ : A = INT(VAL(T$))
20 IF A > 26 or A < 1 THEN GOTO 10
30 A$ = CHR$(64 + a)
40 PRINT A$
RUN
```

***CLEAR***

| This function is not supported. |
|---|

## ***CLOSE***

DEFINITION: Close a RAM file.

FORMAT: CLOSE [#]filenum[,filenum,...]

- Close a file or a series of files

or

CLOSE

- Close all files

filenum - An arithmetic expression with a value between 5 and 255, inclusive.

NOTES:

1. The file must be opened before it can be closed. If no files are opened when a CLOSE all files statement is executed, COMPSEE BASIC ignores the statement.
2. If an error occurs while closing a series of files, all files in the list before the error are closed.

EXAMPLES:

Example 1 - Closes file 10 (assuming file 10 was previously opened).

```
CLOSE #10
```

Example 2 - Closes files 10, 20, and 30 (assuming they were previously opened).

```
CLOSE 10,20,30
```

Example 3 - Closes all opened files.

```
CLOSE
```

## ***CLS***

DEFINITION: Clear the LCD screen and put the cursor in the home position (row 1, column 1).

FORMAT: CLS

NOTES:

EXAMPLE:

```
10  A$ = BARCODE$ : IF A$ = ""  THEN GOTO 10
20  IF A$ = "Z" THEN CLS ELSE PRINT #0, USING "P1,C15",A$;
30  GOTO 10
RUN
```

If BARCODE$ returns a bad read indication, clear the screen; otherwise, print the bar code to the screen.

## ***CMDKEY System Variable***

DEFINITION: Return the identity of the key used to terminate the last keypad input statement (INPUT$ #1...).

FORMAT: x = CMDKEY

NOTES:

1. The possible values are:

   -1 if no INPUT$ #1 statement has been processed;

   0 if the ENTER key ended the last INPUT$ #1 statement;

   1 if the up arrow key ended the last INPUT$ #1 statement;

   2 if the down arrow key ended the last INPUT$ #1 statement.

   If the last INPUT$ #1 statement was terminated by a character (via auto-entry), the ASCII value (in decimal) of that character is returned.

2. A RUN command resets the value of CMDKEY to -1.

EXAMPLE:

```
10 PRINT CMDKEY
20 INPUT$ #1,"E",5,T$
30 PRINT CMDKEY
RUN
```

Line 10 sends a -1 out the RS-232-C port because no INPUT$ statement has been processed. The value printed at line 30 depends on the terminating key for the INPUT$ statement at line 20.

## ***CONT***

DEFINITION: Continue the application program after a STOP or a ALT / C, ALT / D key sequence.

FORMAT: CONT

NOTES:

1. When COMPSEE BASIC processes a STOP statement, a ALT / C, ALT / D key sequence, a breakpoint, or a single step control is passed to DIAGNOSTIC mode. When the DIAGNOSTIC session is terminated, COMPSEE BASIC is put into IMMEDIATE mode. If a CONT command is then issued, the program continues from the point of the break. CONT can only be used in IMMEDIATE mode.
2. DIAGNOSTIC mode can also be entered by an error. If a CONT command is issued after an error-induced DIAGNOSTIC session, a CANNOT CONTINUE error occurs.
3. If the application program is edited before the CONT command is issued, a CANNOT CONTINUE error results.
4. A G command in DIAGNOSTIC mode has the same effect as Quitting the DIAGNOSTIC mode and issuing a CONT command. See the section on DIAGNOSTIC mode for further details.

EXAMPLES:

```
10 FOR I = 1 TO 5
20 PRINT I : IF I = 3 THEN STOP
30 NEXT I
RUN
1
2
3
CONT
4
5
```

When I is equal to 3, execution is stopped and the DIAGNOSTIC mode is entered. After the DIAGNOSTIC session is terminated, a CONT command causes execution to resume from the point of interruption.

### ***COS Function***

This function is not supported.

### ***DATA***

DEFINITION: Store the numeric and string constants to be used by the READ statement.

FORMAT: DATA constant[,constant,...]

constant - Must be a numeric or a string constant; no expressions or variables are allowed.

NOTES:

1. DATA statements are non-executable.
2. In order to speed program execution, all DATA statements should be grouped together at the beginning of the application program. COMPSEE BASIC searches from the beginning of the program for the 1st DATA statement, and from the previous DATA statement for the next DATA statement.
3. Each successive READ accesses the next DATA element. DATA statements may be reread by using the RESTORE statement.

EXAMPLES:

Example 1 - A contains a 1; B contains a 2; and C contains a 3.

```
10 DATA 1,2,3
20 READ A,B,C
RUN
```

Example 2 - An error results; not enough data in the program.

```
10 DATA 5,6
20 READ A,B,C
RUN
```

Example 3 - A contains a 1; B contains a 2; C contains a 3; and D contains a 4.

```
10 DATA 1,2,3,4
20 DATA 5,6
30 READ A,B
40 READ C,D
RUN
```

### ***DATE$ System Variable***

DEFINITION: Set or retrieve the date.

FORMAT:
Set the date = DATE$ = "dstring"
Retrieve the date = x$ = DATE$

dstring - The date string in the form YY/MM/DD, where:

YY - year (e.g., 95)
MM - month (01 for January)
DD - day

NOTES:

1. COMPSEE BASIC always represents the DATE$ string in eight characters; therefore, the year, month, and day are padded with zeros when necessary. For example, January 1, 1996 is represented as 96/01/01. Conversely, the leading zeros are not necessary when settling the date. For example, 96/1/1 is sufficient to set the date to January 1, 1996.
2. Any data after the day is ignored by COMPSEE BASIC (see Example 3).

EXAMPLES:

Example 1 - Stores today's date in variable TDATE$.

```
TDATE$ = DATE$
```

Example 2 - Sets the data to July 14, 1996.

```
DATE$ = "96/7/14"
```

Example 3 - Sets the date to December 25, 1996. The characters after the day are ignored.

```
DATE$ = "96/12/25 XMAS"
```

### ***DAY System Variable***

DEFINITION: Retrieve the day.

FORMAT: x = DAY

NOTES:

1. COMPSEE BASIC returns a number from one to seven according to the following list:
   1 = Sunday
   2 = Monday
   3 = Tuesday
   4 = Wednesday
   5 = Thursday
   6 = Friday
   7 = Saturday
2. COMPSEE BASIC calculates the day based on the date set by the DATE$ system variable. The starting point for the day calculation is January 1, 1984. A year from 0 to 83 is interpreted as the next century. For example, 83/1/1 is interpreted by the DAY system variable to be January 1, 2083.

EXAMPLES:

```
10  DATE$ = "92/8/28"
20  DIM A$*40,B$*40
30  A$ =" SunMonTueWedThrFriSat"
40  B$ =" JanFebMarAprMayJunJulAugSepOctNovDec"
50  TDATE$ = DATE$ : TDAY = DAY
60  MONTH$ = B$(VAL(TDATE$ (4:5))*3:(VAL(TDATE$ (4:5))*3) + 2)
70  WEEKDAY$ = A$(DAY*3:(DAY*3) + 2)
80  PRINT "Today is" + WEEKDAY$ + " " + MONTH$ + " " + TDATE$(7:8) + ",19" + TDATE$(1:2)
    RUN

    Today is Thr Aug 28, 1992
```

This example shows how the DATE$ string and the DAY number can be converted to text. Line 60 uses the month number returned by DATE$ to find the text in B$. Line 70 uses the number returned by DAY to find the text in A$.

### ***DELETE***

DEFINITION: Delete a line or lines of the application program stored in memory.

FORMAT: DELETE start[-end]

start - An application program line number.

end - An application program line number.

NOTES:

1. One line or range of lines can be deleted. For example:
   Delete line 10 = DELETE 10
   Delete lines 10 through 30, inclusive = DELETE 10-30

Unlike the LIST command, the start and end line numbers in the DELETE command must exist in the program. Otherwise an Error 12 will result.

2. The deletion of a large program is a slow process; the Apex II terminal may power down during delete. When the Apex II terminal is turned back on, the delete continues.

EXAMPLES:

For examples 1 and 2, assume that the following program is stored in memory:

```
10 FOR I = 1 TO 5
20 A = A + I
30 NEXT I
40 PRINT A
```

Example 1 - Deletes line 10, 20, and 30.

```
DELETE   10-30
```

Example 2 - Results in an Error 12 because line 35 does not exist.

```
DELETE   20-35
```

## ***DIM***

DEFINITION: Allocate data storage for arrays and strings.

FORMAT: DIM variable [,variable,variable,...]

variable - An array or string variable. Variable can be in nine forms:

1. A(X)
   One-dimensional numeric array where X is the maximum element.
2. A(X,Y)
   Two-dimensional numeric array where X is the maximum row and Y is the maximum column.
3. A$(X)
   One-dimensional string array with a default maximum element length of 18 characters.
4. A$(X,Y)
   Two-dimensional string array with a default maximum element length of 18 characters.
5. A$(X)*20
   One-dimensional string array with a user-defined maximum element length of 20 characters. The maximum element length can be up to 255 characters.
6. A$(X,Y)*30
   Two-dimensional string array with a user-defined maximum element length of 30 characters.
7. A$*200
   String scalar with a user-defined maximum length of 200 characters.

8. A$
   (OPTIONAL) - String scalar with a maximum length of 18 characters.
9. A
   (OPTIONAL) - Numeric scalar.

NOTES:

1. A variable can only be dimensioned once in an application program.
2. Scalar variables (e.g., A$ and A) are dimensioned the first time they are used in a COMPSEE BASIC statement. Arrays must be dimensioned in a DIM statement.
3. The maximum number of elements an array can have is 65535. A one-dimensional array can have a maximum index of 65534. The 0 element is counted because COMPSEE BASIC uses option base 0. For more information on arrays, see the ARRAYS instruction in this section of the manual.
4. When a variable is dimensioned by a DIM statement or through its first use, its initial value is zero for numerics and null for strings.
5. The DIM statement does not accept numeric expressions for the string length parameter. Numeric expressions are allowed for the dimension parameters.

EXAMPLE:

```
DIM A$*200,B$(5),C(10,10)
```

Defines a string scalar with a maximum of 200 characters, a string array with 6 elements each having a maximum length of 18 characters and a numeric array with 121 elements (remember the 0 row and the 0 column are allocated data space).

## ***DIR***

DEFINITION: Send the status of the Apex II terminal memory out the RS-232-C port.

FORMAT: DIR ["filename"]

filename - The name of an existing file; including the filename gives the status of only that file.

NOTES:

1. The output of DIR is:

```
FILENAME     #  PNTR  LNTH  CNTR  START  STOP
filen      num  point rlen  count saddr  eaddr

PROGRAM         DATA         FILE          FREE
programl        dataleng           fileleng          freespce
```

filen - The name of a RAM file.
num - The number the file is opened under. If the file is closed, num is zero.
point - The number of the next record. If the file is closed, point is zero.
rlen - The length of each record in the file.
count - The number of records in the file.
saddr - The start address of the file in memory (in hex).

eaddr - The end address of the file in memory (in hex).
programl - The number of bytes in the application program.
dataleng - The number of bytes used for variable and array storage.
freespce - The number of unused bytes in memory.
fileleng - The total number of bytes used by all RAM files.

EXAMPLES:

Example 1 - File TESTA is opened as file number 30, points to record 1; has 123 records, each of which is 10 bytes long, and starts at address 1FFFFH and ends at address 1F800H in memory. File TESTB is closed, has 5 records, each of which is 100 bytes long, and starts at address 1F7FFH and ends at address 1F000H in memory. The application program is 128 bytes long, uses 18 bytes of memory for data storage, and uses 4096 bytes for file storage.

```
DIR

FILENAME   #    PNTR  LNTH  CNTR  START  STOP
TESTA      030  00001 0010  00123 1FFFF  1F800
TESTB      000  00000 0100  00005 1F7FF  1F000

PROGRAM         DATA         FILE         FREE
00000128        00000018     00004096     00076236
```

Example 2 - This reports the status of file TESTA only.

```
DIR      "TESTA"

FILENAME   #    PNTR  LNTH  CNTR  START  STOP
TESTA      030  00001 0010  00123 1FFFF  1F800

PROGRAM         DATA         FILE         FREE
00000128        00000018     00002048     00078284
```

***END***

DEFINITION: Power down the Apex II terminal.

FORMAT: END [,RESUME]

NOTES:

1. The optional RESUME causes the program to continue with the statement after the END statement when the Apex II terminal is powered up. An END without the RESUME restarts the program.
2. The RESUME option has no effect in IMMEDIATE mode.

EXAMPLE:

```
10  CLS
20  TMP$ = TIME$
30  IF TMP$<>TIME$ THEN PRINT #0, USING"P1,C10",TIME$; :TMP$=TIME$
40  IF TMP$ = "18:00:00" THEN END,RESUME
50  GOTO 30
```

The time is continuously displayed on the screen. At six o'clock PM, the Apex II terminal powers down. When the unit is powered back up, the first statement processed is the GOTO 30.

## ***EOF Function***

DEFINITION: Return the status of the RS-232-C receive buffer, keyboard buffer, or a RAM file.

FORMAT: x = EOF(fnum)

fnum -An arithmetic expression with the following values:

| | | |
|---|---|---|
| 1 | = | Keyboard buffer |
| 4 | = | RS-232C receive buffer |
| 5-255 | = | RAM file |

NOTES:

1. When using EOF on a buffer (fnum = 1 or 4), -1 is returned if the buffer is empty; a 0 is returned if the buffer contains characters.
2. When using EOF on a RAM file, -1 is returned if the file pointer is at the end of the file; 0 is returned otherwise.

EXAMPLES:

Example 1 - The WHILE/WEND loop is processed as long as the receive buffer contains characters.

```
10 WHILE EOF(4) <> -1
20 INPUT$ #4,1,T$
30 LINE$ = LINE$ + T$
40 WEND
50 PRINT T$
```

Example 2 - The "TEST" file is read until no more records exist. The use of the EOF function in this manner prevents a READ PAST END OF FILE error.

```
10 OPEN "TEST" AS #10
20 WHILE EOF(10) <> - 1
30 READ #10, USING "C10",T$
40 PRINT T$
50 WEND
51
```

### ***ERR and ERL System Variables***

DEFINITION: Return the error number and the line number of the error.

FORMAT: X = ERR
and
Y = ERL

NOTES:

1. A RESUME statement resets ERR and ERL to zero.
2. If the error occurred in IMMEDIATE mode, ERL is not affected (i.e., ERL contains the previous error line number of a zero).

EXAMPLE:

```
10 ONERR GOTO ERRH
20 SIGNAL 12
30 STOP
40 '
50 ERRH: PRINT ERR,ERL
```

The error number (ERR) is 14 and the error line (ERL) is 20. They are printed out the RS-232-C port.

### ***EXP Function***

This function is not supported.

### ***FILE$ System Function***

DEFINITION: Return the RAM file name associated with a file number.

FORMAT: x$ = FILE$(fnum)

fnum -An arithmetic expression with a value between 5 and 255, inclusive.

NOTES:

1. If fnum is not associated with a file, FILE$ returns a null string.
2. The file name is right-padded with blanks to eight characters.

EXAMPLE:

```
10 OPEN "AAA" AS #10 LEN = 20
20 A$ = FILE$(10) : B$ = FILE$(20)
RUN
```

A$ contains "AAA" plus five blanks and B$ contains a null string.

## ***FOR and NEXT***

DEFINITION: Establish a loop to perform a set of statements a given number of times.

FORMAT: FOR variable = start TO stop [STEP increment]

NEXT [variable[,variable]...]

variable - A numeric scalar variable which serves as the loop counter.

start - An arithmetic expression whose value is the initial value of the counter.

stop - An arithmetic expression whose value is the final value of the counter.

increment - An arithmetic expression whose value is added to the counter. If STEP is not included, the default increment is one.

NOTES:

1. The statements after the FOR statement are executed until the NEXT statement. The increment value is then added to the counter.
2. *Positive increment values:* If the counter is less than or equal to the stop value, the loop continues; otherwise, the loop terminates and the statement after NEXT executes.
3. *Negative increment values:* If the counter is greater than or equal to the stop value, the loop continues; otherwise, the loop terminates and the statement after NEXT executes.
4. COMPSEE BASIC checks the start and stop values before executing the loop for the first time. If the start value is greater than the stop value and the increment is positive, the loop is skipped and COMPSEE BASIC processes the statement after the NEXT statement. The same is true if the start value is less than the stop value and the increment is negative. An increment value of zero results in an infinite loop.
5. FOR-NEXT loops can be nested. If the loops are nested, each loop must have its own variable counter. Each loop may be terminated by a specific NEXT (NEXT I) or a generic NEXT (NEXT). If generic NEXTs are used, COMPSEE BASIC matches the NEXT statement with the most recent FOR statement (see Example 2). If the loops have the same end point, one NEXT statement can be used. For example, NEXT J : NEXT I is equivalent to NEXT J,I (see Example 3).

Loops are stack-driven. This allows an unlimited number of loops within a program. The only restriction is that the loop nest cannot exceed the system limit (see ERROR MESSAGES, Section 17, under Error 2 for the system limit). A nest is a loop that is processed within another loop.

For instance, this program has a nest value of three:

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO 3
30 FOR K = 1 TO 5
40 PRINT I + J + K
50 NEXT K,J,I
```

A stack scheme of processing loops requires that all loops be resolved; i.e., the loop counter must be greater than the loop end value (or the loop counter must be less than the loop end value if the loop is in the negative direction; for instance: FOR I = 10 TO 1 STEP -1). The first example must be changed to:

```
10 FOR I = 1 TO 1000
20 IF INKEY$ = "Q" THEN I = 1000 ELSE PRINT I
30 NEXT I
40 EXIT:
```

In this example, the loop counter (I) is set to the end value. The next time the NEXT statement is processed, the loop terminates naturally.

EXAMPLES:

Example 1 - Sends 1, 2, 3, 4, and 5 out the RS-232-C port.

```
10 FOR I = 1 TO 5
20 PRINT I
30 NEXT I
```

Example 2 - Notice how the generic NEXT matches with the most recent unmatched FOR. The NEXT in line 40 matches with the FOR in line 20 and the NEXT in line 60 matches the FOR in line 10.

```
10 FOR I = 1 TO 3
20 FOR J = 1 TO 2
30 PRINT I,J
40 NEXT
50 A = A + J
60 NEXT
RUN
1  1
1  2
2  1
2  2
3  1
3  2
```

Example 3 - Since both loops terminate at the same point, only one NEXT need be used.

```
10 FOR I = 1 TO 2
20 FOR J = 1 TO 2
30 PRINT I,J
40 NEXT J,I
RUN
1  1
1  2
2  1
2  2
```

Example 4 - Causes an error. NEXT I,J is equivalent to NEXT I : NEXT J. The inner loop (variable J) must terminate before the outer loop (variable I). A NEXT with no FOR error results.

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO 7
30 PRINT I,J
40 NEXT I,J
RUN
```

Example 5 - The stop value is calculated after the start value. In some BASICs, the stop value has a value of 12.

```
5  I= 10
10 FOR I = 1 TO I + 2
20 PRINT I
30 NEXT I
RUN
1
2
3
```

***NOTE***

```
You must satisfy a FOR/NEXT loop before exiting; otherwise,
you will receive an Error 2.
```

EXAMPLE:

```
FOR X = 1 to 400
  .
  . 1 more code here
  .
  4 Bailout = 1 then x = 400;proper way to fast exit a loop
Next X
```

## ***Form String***

Formatted I/O operations (READ, WRITE, REWRITE and formatted PRINT) are performed via a form string. The string tells COMPSEE BASIC how to interpret each piece of data in the data list. Every data field must have an associated FORM field or an error results.

Example 1 - a formatted PRINT

```
PRINT #4,USING "C11,N5.0","HELLO THERE",900
```

The FORM string is enclosed between quotation marks. FORM fields are separated by commas. In example 1, C11 and N5.0 are form fields; "HELLO THERE" and 900 are the associated data fields.

The FORM field is comprised of two parts: a data type followed by a field length. The data type is denoted by a single character. In example 1, C and N are data types; 11 and 5.0 are field lengths.

COMPSEE BASIC has 6 FORM fields (listed in data type order):

1. **C - character**
   The length range is from 1 to 255, inclusive. If the data field is larger than the FORM field, an error results. If the data field is smaller than the FORM field, COMPSEE BASIC pads with blanks to the end of the FORM field.

   Example 2 - blank padding

   ```
   DIM A$*20
   'System defaults to 18 characters
   A$ = "THIS IS A GOOD WRITE"
   WRITE #10,USING "C30", A$
   'Pad with 10 blanks
   ```

   Example 3 - ERROR: data field too large

   ```
   WRITE #10,USING "C3", "BAD WRITE"
   ```

2. **E - exponential**
   The length is expressed in the form X.Y where X is the total field length and Y is the number of significant decimal places. X must be at least seven greater than Y to account for the sign, the mantissa, the decimal point, a blank, an "E", the exponent sign, and the exponent. Since COMPSEE BASIC stores eight significant digits, the Y maximum is 7 (the eighth digit is used for the mantissa). The maximum total field length is 255. The sign for the mantissa is a minus sign for negative numbers and blank for positive numbers. The sign for the exponent is a minus sign for negative exponents and a plus sign for positive exponents. If the data field is less than the total field length, COMPSEE BASIC pads with blanks. If the data field has more decimal places than the FORM field allows, COMPSEE BASIC truncates the extra decimal places. The E FORM field is used differently depending on whether the operation is an input or an output. For a formatted input (READ), the Y value has no significance.

Example 4 -formatted READ

```
READ #10,USING "E15.7",A
```

In Example 4, the record pointed to by the read pointer in file #10 is read. The first 15 bytes of that record are to be interpreted as numeric data (0 through 9, +, -, or E). The 15 bytes can be in decimal or exponential form. If none of the first 15 bytes are numeric, an error results. For a formatted output (PRINT, WRITE, REWRITE), the Y value is used.

Example 5 - valid PRINT

```
A = -9.23
PRINT #4,USING "E15.2",A
```

The result in Example 5 is -9.23 E 0 followed by six blanks.

Example 6 - ERROR: field too small

```
A = 1.1 E10
WRITE #10,USING "E8.1",A
```

In order to WRITE A, the total field length must be at least 9.

Example 7 - truncation

```
PRINT #4,USING "E10.2", -99.23568
```

The result in Example 7 is -9.92 E + 1 followed by one blank.

**3. N - numeric**

The length is expressed in the form X.Y where X is the total length and Y is the number of decimal places. The X value must always take the sign byte into account. If the Y value is 0 (i.e., no decimal places), the X value must be at least two; a sign byte and a digit byte. If the Y value is greater than 0 (i.e., decimal places are desired), the X value must also take the decimal point into account. The maximum value for Y is eight (COMPSEE BASIC stores eight significant digits). The maximum value for X is 10 if Y is greater than 0 and 9 if Y is equal to 0. If the data field is less than the FORM field, COMPSEE BASIC right-justifies. If the data field is greater than the FORM field, an error results. If the number of decimal places is greater than the Y value, COMPSEE BASIC truncates the extra decimal places. The N FORM field (like the E FORM field) is used differently depending if the operation is an input or an output. For a formatted input (READ), the Y value has no significance. The sign for the mantissa is a minus sign for negative numbers and blank for positive numbers.

Example 8 - valid PRINT

```
PRINT #4,USING "N6.2",-9.23
```

The result in Example 8 is (blank)-9.23

Example 9 - ERROR: data field too large

```
A = 9854.23
PRINT #4,USING "N5.2",A
```

The largest number that can be written with an N5.2 field is 9.99. The sign is one byte, the decimal point is one byte, and the decimal places are two bytes; only one byte is left for the number.

Example 10 - truncation

```
A = 9.3456
PRINT #4,USING "N6.2",A
```

The result is (blank) (blank for the sign)9.34

**4. P - position**

The P FORM field is unlike the previous fields in that it is not matched with a data field. The P field positions the pointer within a record (for READ, WRITE, and REWRITE) or positions the cursor (for a formatted PRINT). The length tells COMPSEE BASIC where to position; the length range is 1 to 2047, inclusive. The maximum length when operating on RAM files depends on the record length of the RAM file.

In a formatted PRINT to the RS-232-C port, if the length is less than the present cursor position, a carriage return is written first. Blanks are sent out the RS-232-C port up to the position where the cursor is to be placed. In a formatted PRINT to the screen, if the length is greater than 32 and less than 2047, the cursor is placed in row 2 column 17 (i.e., off the screen). The screen has 32 positions -2 rows of 16 columns.

Example 11 - valid PRINT

```
PRINT #0,USING "P2,C3","JOE"
```

The screen (field #0) cursor is put at position 2 (first row, second column) and the "J" is written at position 2.

Example 12 - valid REWRITE

```
REWRITE #10,USING "P5,C1",REC = 1,"K"
```

The record pointer is moved to the fifth position of record one and a K is written in position 5.

Example 13 - valid PRINT

```
PRINT #4,USING "C4,P1,C4","ABEL","CAIN"
```

A carriage return is written to the RS-232-C port between "ABEL" and "CAIN".

Example 14 - valid PRINT

```
PRINT #0,USING "C4,P54","ABEL";
```

After printing "ABEL", the cursor is moved off the screen.

**5. W - internal numeric format**

COMPSEE BASIC converts all numbers to a six byte notation:

exponent byte

| | | | |
|---|---|---|---|
| 80H | | = | E 0 |
| 81H | | = | E + 1 |
| 82H | | = | E + 2 and so on |
| 7FH | | = | E -1 |
| 7EH | | = | E -2 |
| 7DH | | = | E -3 and so on |
| sign byte | 1 | = | negative |
| | 0 | = | positive or zero |
| data byte | 3 | = | 32 bit HEX value |
| data byte | 2 | = | 32 bit HEX value |
| data byte | 1 | = | 32 bit HEX value |
| data byte | 0 | = | 32 bit HEX value |

For example, -82.316 is stored as 7DH 01H 00H 01H 41H 8CH and .00234 is stored as 7BH 00H 00H 00H 40H EAH. The W FORM field stores the number in internal form. The W field is invalid for the formatted print; it can only be used with RAM files. The length must always be six.

Example 15 - valid WRITE

```
WRITE #10,USING "W6",9
```

Example 16 - ERROR: cannot use W field with a PRINT

```
PRINT #4,USING "W6",9
```

**6. X - skip**

As with the P FORM field, the X field has no data field associated with it. The X field writes blanks to positions for a WRITE, a PRINT to the RS-232-C port and a REWRITE. The X field skips positions for a READ and a PRINT to the screen. The length tells COMPSEE BASIC how many positions to skip or blanks to write; the length range is 0 to 2047.

Example 17 - valid PRINT

```
PRINT #0,USING "P1,X5,C3","JOE"
```

The cursor is moved to position 1 (first row, first column), five positions are skipped and the "J" is written in the sixth position. Note the difference between P5 and X5 (assume the cursor is at position 1). P5,C1 puts the character in the fifth column; X5,C1 puts the character in the sixth column.

Example 18 - valid READ

```
READ #10,USING "X5,C1",A$
```

The first five positions of the record are skipped and the character is READ from the sixth position. Note the difference between X5 and P5. P5 skips to the fifth position and the character is read from the fifth position. X5 skips five positions and reads the character from the sixth position.

## ***FREC Function***

DEFINITION: Searches an open file for the next occurrence of a specific key field (string of characters).

FORMAT: RECNUM=FREC(KEY$,KSTART,KLEN,FILENUM,STARTREC)

RECNUM The record number containing the next occurrence of the key field.

- >0 = record number
- 0 = key not found
- <0 = error
  - -1 = Invalid record number
  - -2 = Invalid file number
  - -3 = File not found
  - -4 = Record starts past the end of the file
  - -5 = Invalid key length
  - -6 = Invalid key start position
  - -7 = Invalid key string length
  - -8 = Invalid number of parameters

KEY$ - Key field to match (1 to 30 characters)
KSTART - Start position in each record (character string) of key field
KLEN - Length of key field (1 to 30 characters)
FILENUM - File to search
STARTREC - Record number to begin the search

***NOTE***

A search of an empty file returns an error -4.

## ***FREESP System Variable***

DEFINITION: Return the amount of unused memory.

FORMAT: X = FREESP

NOTES:

1. The number returned by FREESP is the total amount of memory in the Apex II terminal minus the file space minus the amount used for variable storage.

EXAMPLE:

```
PRINT FREESP
```

This command sends the amount of unused memory out the RS-232-C port.

***NOTE***

The operating system allocates memory in increments of 2K. Therefore if a file is on a boundary, FREESP may change by increments of 2048 bytes.

## ***GOSUB and RETURN***

DEFINITION: Transfer program execution to a subroutine.

FORMAT:

```
GOSUB line number        GOSUB label
.                           .
.          OR               .
.                           .
RETURN                   RETURN
```

line number - Application program line number which is the first line of the subroutine.

label - Application program label which is the first line of the subroutine. See the LABEL section of this manual for further information on labels.

NOTES:

1. The RETURN statement returns program execution to the first statement after the GOSUB. A subroutine may have more than one RETURN; however, care must be taken so that a RETURN will not be executed without a corresponding GOSUB.
2. Subroutines can be nested; i.e., a subroutine can call another subroutine. The system limit for nested routines is 16.

3. COMPSEE BASIC does not support recursive subroutines. The program below can put the Apex II terminal into an unpredictable state:

```
10  ONBAR GOSUB GETBAR
20  MAIN:  GOSUB GETBAR
30  IF A$ = "" THEN GOTO MAIN
40  IF A$ = "Z" THEN PRINT "Unable to decode" ELSE PRINT A$
50  GOTO MAIN
100 GETBAR:  A$ = BARCODE$
110 BARCNT = BARCNT + 1
120 RETURN
```

Recursion occurs if COMPSEE BASIC is interrupted (see ONBAR) while the GETBAR routine executes.

EXAMPLE:

```
10   IF A = 1 THEN GOSUB TRUE ELSE GOSUB 1000
20   STOP
30   '
500  TRUE: PRINT "TRUE": RETURN
900  '
1000 PRINT "FALSE"
1010 RETURN
```

If A equals 1, subroutine TRUE executes and "TRUE" prints out the RS-232-C port. If A does not equal 1, the subroutine at line 1000 executes and "FALSE" prints out the RS-232-C port.

## ***GOTO***

DEFINITION: Branch program execution to a line.

FORMAT: GOTO line number
or
GOTO label

line number - Application program line number.

label - Application program label. See the LABEL section of this manual for further information on labels.

NOTES:

1. If execution is transferred to a non-executable statement (e.g., a remark or a DATA statement), execution starts at the next executable statement.
2. A GOTO statement issued from IMMEDIATE mode starts execution of the application program from the specified line or label. If the line or label does not exist, an error results. The GOTO statement is NOT equivalent to a RUN command. A RUN command initializes the application program by resetting all variables and pointers; a GOTO statement preserves all variables and pointers.

EXAMPLES:

```
10   IF A = 1 THEN GOTO TRUE ELSE GOTO 1000
20   '
500  TRUE: PRINT "TRUE" : STOP
900  '
1000 PRINT "FALSE"
1010 STOP
```

If A equals 1, the line with label TRUE is executed and "TRUE" is printed out the RS-232-C port. If A does not equal 1, line 1000 is executed and "FALSE" is printed out the RS-232-C port.

## ***IF***

DEFINITION: The IF statement lets you evaluate a condition and works together with the THEN clause to take a cause of action based on the evaluation.

FORMAT: IF test THEN clause [ELSE clause]

test - Any expression that resolves to a true (not 0) or a false (0).

Examples are:

IF 5
IF A = 6
IF B$ <> "JOE"
IF ABEL AND (CAIN$="SETH")

IF "test" is invalid because "test" does not evaluate to a true or a false.

clause- Any COMPSEE BASIC statement or series of COMPSEE BASIC statements (separated by colons). If the result of the IF test is true, the THEN clause is processed. If the result of the IF test is false, the ELSE clause is processed. If the ELSE clause is not present in RUN mode, the next line number is processed; in IMMEDIATE mode, execution is terminated.

NOTES:

1. IF statements can be nested. The only limitation is the length of the line. (Length maximum = 255 characters.)
2. All IF statements must have a THEN clause. However, ELSE clauses are optional. If a nested IF statement has the same number or less ELSE clauses than THEN clauses, each ELSE is matched with the closest unmatched THEN (see Example 2). If the statement has more ELSE clauses than THEN clauses, an error results.
3. The number of GOSUBs which can be nested from within an IF-THEN-ELSE statement is limited by COMPSEE BASIC (see Error Descriptions: Error 2 for this number). The following program has an IF-THEN-ELSE nest of two:

```
 10  IF A <> 1 THEN GOSUB TEST:PRINT A ELSE PRINT B
 20  STOP
100  TEST:IF A = 2 THEN GOSUB TEST1:PRINT C ELSE PRINT D:RETURN
200  TEST1:PRINT E:RETURN
```

EXAMPLES:

Example 1 - Prints the number 5 out the RS-232-C port.

```
A = 1
IF A = 1 THEN PRINT 5 ELSE PRINT 6
```

Example 2 - Prints the number 3 out the RS-232-C port. Note how the ELSE clauses match up with the THEN clauses. If A is 0, the number 4 is printed out the RS-232-C port.

```
10  A = 1 : B = 0 : C$ = "JOE"
20  IF A THEN IF A = 1 THEN IF C$ = "ABEL" THEN PRINT 1:PRINT 2 ELSE PRINT 3 ELSE PRINT 4
```

Example 3 - No numbers are printed. The first THEN clause in the nest has no associated ELSE clause. The next statement processed is the next line number. In this example, line 20 is the last line; therefore, the program ends.

```
10  A = 0 : B = 5 : C$ = "CAIN"
20  IF A <> 0 THEN IF B THEN IF C$ = "CAIN" THEN PRINT 1 PRINT 2 ELSE PRINT 3:PRINT 4 ELSE
PRINT 5
```

Example 4 - Causes an error. If A does not equal 1, "ELSE" is printed out the RS-232-C port. Line 20 is then processed and a NEXT with no FOR error results. If A equals 1, the value of I is printed out the RS-232-C port. COMPSEE BASIC then encounters the ELSE clause and skips to the next line. Line 20 is processed, the loop continues and the second value of I is printed out the RS-232-C port. COMPSEE BASIC does not see the THEN clause and flags the ELSE clause as having no associated THEN.

```
10  IF A = 1 THEN FOR I = 1 TO 3: PRINT I ELSE PRINT "ELSE"
20  NEXT I
```

## ***INKEY$ System Variable***

DEFINITION: Read a single character from the keypad buffer.

FORMAT: x$ = INKEY$

NOTES:

1. A null string is returned if the keypad buffer is empty.
2. The character read from the keypad buffer is not echoed to the screen.
3. A ALT / C breaks the program execution; INKEY$ does not return the ALT / C character.

EXAMPLES:

```
10 A$ = INKEY$:IF A$ = ""  THEN GOTO 10
20 IF A$ = "Q" THEN STOP ELSE GOTO 10
RUN
```

Line 10 puts the program into a loop until a key is pressed. If the key pressed is a "Q", the program stops; otherwise, the program goes back into the loop.

## ***INP System Variable***

DEFINITION: Return the value of certain system bit flags.

FORMAT: x = INP

NOTES:

1. INP returns a number between 0 and 255 which corresponds to the following flags:

   1 = RTS
   2 = STOP_MOTOR
   4 = NO_SCAN_BIT
   8 = DST
   16 = NO_RELOAD
   32 = NO_ALT_C
   64 = Unused
   128 = CTS

| Flag | If set | If clear |
|---|---|---|
| RTS | RTS RS 232-C control line is not asserted | RTS RS 232-C control line is asserted |
| STOP_MOTOR | The mirror will not reciprocate during scan | The mirror will reciprocate during a scan |
| NO_SCAN_BIT | Scanning is disabled | Scanning is enabled |
| DST | Automatic clock compensation for daylight savings time is enabled | Automatic clock compensation for daylight savings time is disabled |
| NO_RELOAD | Power down timer is not reloaded at the beginning of every COMPSEE BASIC statement | Power down time is reloaded at the beginning of every COMPSEE BASIC statement |
| NO_ALT_C | The INPUT$ #4 statement accepts a ALT / C as a character | The INPUT$ #4 statement aborts upon receiving a ALT / C character |
| CTS | CTS RS232-C control line is not asserted | CTS RS232-C control line is asserted |

2. In order to determine whether a flag is set or clear, you should AND the value of INP with the flag s associated number from the table above. For example, the following line determines if the STOP _ MOTOR flag is set or clear: IF (INP AND 2) = 2 THEN FLAGSET$ = "YES" ELSE FLAGSET$ = "NO"
3. The unused flags return cleared.
4. The STOP _ MOTOR, NO _ SCAN _ BIT, NO _ RELOAD, NO _ ALT _ C and DST flags are all clear at initial power up.
5. The flags can be set or cleared using the OUT statement.
6. The NO _ RELOAD flag relates to COMPSEE BASIC s power down timer. Through the SYSPARMS statement, the user can set a timer value of 0 to 255 seconds. This timer causes the unit to power down after the specified number of seconds of inactivity. The timer is normally reset under these conditions: After a key is pressed on the Apex II terminal keypad; before the execution of a COMPSEE BASIC statement; or when the RS-232-C port is used (see the SYSPARMS statement for more information on the power down timer).

   If the NO _ RELOAD flag is set, the power down timer is not reset before a COMPSEE BASIC statement. This allows the user to continue processing an application program while waiting for data to be entered. The Apex II terminal will power down if the data is not entered in the specified amount of time. Once data is entered through any of the following means, the NO _ RELOAD flag is cleared and the power down timer is reset after the conditions described above.

EXAMPLE:

```
IF (INP AND 128) = 128 THEN PRINT #0, USING "P1,
C8", "END COMM"
```

The CTS control line is checked. If the line is not asserted, RS-232-C communications has been terminated.

### ***INPUT$***

DEFINITION: Receive input from keypad buffer or the RS-232-C receive buffer.

FORMAT: INPUT$ #filenum [,"options"],length,variable

filenum - An a.e. with a value of 1 or 4
1 = keypad buffer
4 = RS-232-C receive buffer

options - This parameter is used for keypad input only. Input from the RS-232-C receive buffer ignores this parameter. The options must be capital letters:

"E" - Auto-ENTER
Once the length number of characters are entered or the cursor is moved to the last position by the right arrow key, an ENTER is automatically issued and the INPUT$ is terminated.

"I" - INVISIBLE characters
The characters are not echoed to the unit's screen. If the INPUT$ has default data on the screen, the screen is overwritten with blanks. See Note 2 for more information on default data.

"T" - TRUNCATE padding
An input string is normally right padded with blanks so the string length is equal to the length parameter (see below). This option suppresses the padding.

length - The number of characters INPUT$ expects to receive. The length must be between 1 and 255, inclusive.

variable - A character variable or character array element.

NOTES:

1. The INPUT$ options can be in any order.
2. The following information pertains to the keypad buffer only:

   INPUT$ is terminated by the ENTER character (ASCII code 13), the UP ARROW character (ASCII code 30), or the DOWN ARROW character (ASCII code 31). INPUT$ is also terminated by the auto-enter option ("E"). Termination short of length characters right pads the variable with blanks (unless the "T" option is invoked).

   The DEL character (ASCII code 127) deletes the character the cursor is presently on. The BACKSPACE character (ASCII code 8) backs the cursor up and deletes the previous character. The ALT-DEL key defaults to the BACK SPACE character.

   The LEFT ARROW character (ASCII code 29) and the RIGHT ARROW character (ASCII code 28) cause nondestructive cursor movement.

   If an attempt is made to add a character past the end of the input field, assuming the "E" option is not invoked, the last character in the field is overwritten. For example:

   ```
   INPUT$ #1,10,A$
   ```

   This inputs the string "1234567890". If an "L" is then entered, the string becomes "123456789L".

INPUT$ allows for a default input string. The default string is the data on the screen from the start of the INPUT$ to the end of the screen, or from the start of the INPUT$ to the end of the INPUT$, whichever is smaller. For example, the first line of the screen reads:

```
123456789ABCDEFG
```

Consider these statements: LOCATE 1,1 : INPUT$ #1,10,A$

If these statements are processed, the string "123456789A" is the default string (start of the INPUT$ is screen position 1; end of the INPUT$ is position 10).

Consider these statements: LOCATE 1,1 : INPUT$ #1,18,A$

If these statements are processed, the string "123456789ABCDEFG" is the default string (start of the INPUT$ is screen position 1; end of the screen is position 16). A$ contains the default string plus two blanks to pad the string length to 18 characters.

3. A <ALT><C> character (ASCII code 3) terminates an INPUT$ without storing any of the string in the variable. The previous contents of the variable are unchanged.

   The <ALT><C> termination for an RS-232-C INPUT$ statement (INPUT$ #4...) can be disabled using the OUT statement. See the OUT statement for further details.

EXAMPLES:

Example 1 - This takes 10 characters from the keypad buffer and puts them in the string variable A$. The characters are not echoed to the screen and the string is not blank padded.

```
INPUT$ #1,"IT",10,A$
```

Example 2 - This takes 15 characters from the RS-232-C receive buffer and puts them in the string variable S$.

```
INPUT$ #4,15,S$
```

## ***INSTR Function***

DEFINITION: Return the position of a substring within a string.

FORMAT: x = INSTR (spos,string,substring)

spos- An arithmetic expression indicating the starting position of the search.
string - A character string whose value has a length of 0 to 255, inclusive.
substring - A character string whose value has a length of 0 to 255, inclusive.

NOTES:

1. If the length of the substring is greater than the string or the substring is not found, a zero is returned.
2. If the substring is a null string, the starting position (spos) is returned.

EXAMPLES:

Example 1 - Sends a 3 to the RS-232-C port.

```
10 P = INSTR (1,"ABCDEFG","C")
20 PRINT P
RUN
```

Example 2 - A string from the RS-232-C buffer is assigned to variable S$. If any part of the S$ is found in the string N$, the word "FOUND" is printed to the RS-232-C port. If S$ is not found, the message "NOT FOUND" is printed.

```
10 N$ = "GEORGE WASHINGTON"
20 INPUT$ #4,5,S$
30 P = INSTR (1,N$,S$)
40 IF P > 0 THEN PRINT "FOUND" ELSE PRINT "NOT FOUND"
50 STOP
```

### ***INT Function***

DEFINITION: Return the integer portion of an arithmetic expression.

FORMAT: x = INT (num)

num - An arithmetic expression with a valid value as described in the number section of this manual.

NOTES:

1. INT returns the next lowest integer. For example, INT (3.99) returns a 3 and INT (-3.99) returns a -4.

EXAMPLE:

```
10 A = 3.42
20 B = 7.89
30 PRINT INT(A),INT(B)
RUN
```

This sends a 3 and a 7 to the RS-232-C port.

### ***KILL***

DEFINITION: Delete a RAM file.

FORMAT: KILL filename

filename - A character string with length between 1 and 8, inclusive. Any ASCII character between codes 35 and 126, inclusive, is valid.

NOTES:

1. The file must exist in the directory.
2. The file can be opened or closed when it is deleted.
3. KILL "***" deletes all RAM files.

EXAMPLE:

```
KILL "TEST"
```

File "TEST" is deleted (assuming it existed in the directory).

## ***LABEL$ System Variable***

DEFINITION: Return the last label name processed.

FORMAT: x = LABEL$

NOTES:

1. LABEL$ is most useful for error trapping (see Example 1).
2. If an error is pending, the value of LABEL$ is the last label processed before the error. Any labels processed between the occurrence of the error and the clearing of the error condition by a RESUME statement have no effect on LABEL$ (see Example 2).

EXAMPLES:

Example 1 - Line 20 opens file "TEST". If "TEST" does not exist, the file is created in line 100.

```
10  ONERR GOTO 100
20  OPENIT:  OPEN "TEST" AS #10
    .
    .
    .
100 IF ERR = 44 AND LABEL$ = "OPENIT" THEN OPEN "TEST" AS #10 LEN = 20
110 RESUME NEXT
```

Example 2 - Example 2 is the same as Example 1 except the error handling section has a label (ERRH). Because an error is pending, LABEL$ is not updated with the label ERRH. Once the RESUME is executed, LABEL$ updating is re-enabled. At line 30, the value of LABEL$ is "MAIN".

```
10  ONERR GOTO ERRH
20  OPENIT:  OPEN "TEST" AS #10
30  MAIN:
    .
    .
    .
100 ERRH:  IF ERR = 44 AND LABEL$ = "OPENIT" THEN OPEN "TEST" AS #10 LEN = 20
110 RESUME NEXT
```

## ***LEN Function***

DEFINITION: Compute the length of a string.

FORMAT: x = LEN (string)

string - A character expression whose value has a length between 0 and 255, inclusive.

NOTES:

1. The LEN function returns an integer value between 0 and 255, inclusive.

EXAMPLES:

Example 1 - A contains 5, the length of the string A$.

```
10 A$ = "RALPH"
20 A = LEN (A$)
RUN
```

Example 2 - This sends a string of 10 X s (XXXXXXXXXX) to the RS-232-C port.

```
10 I = 10
20 WHILE LEN (A$) < I
30 A$ = A$ + "X"
40 WEND
50 PRINT A$
RUN
```

## ***LET***

DEFINITION: Assign a value to a variable.

FORMAT: [LET] variable = expression

variable - A scalar or an array element.

expression - An arithmetic expression or a character expression. The type of expression must match the type of variable.

NOTES:

1. The LET keyword is optional. The equal sign is sufficient to show the assignment.

EXAMPLE:

```
 5  DIM C$(2,3),D(5)
10  A$ = "HELLO"
20  LET B = 9.876
30  LET C$(2,3) = "GOODBYE"
40  D(0) = 65
RUN
```

The scalar A$ contains "HELLO", the scalar B contains 9.876, the element C$(2,3) contains "GOODBYE", and the element D(0) contains 65.

## ***LIST***

DEFINITION: Send the application program or a RAM file out the RS-232-C port.

FORMAT: LIST [[start] - [end]]
or
LIST "filename"

start - An application program line number.

end - An application program line number.

filename - c.s. which represents a valid file name.

NOTES:

1. The entire application program or parts of the program can be listed. For example:

```
LIST 10          List line 10.
LIST 10-30  List lines 10 through 30, inclusive.
LIST 35-    List from line 35 to the end of the program.
LIST -40    List from the start of the program to line 40.
LIST        List the entire program.
LIST -           List the entire program.
```

   COMPSEE BASIC ignores any line numbers that do not exist in the application program. If the non-existent line number is the start of a LIST range (e.g., LIST 27-40 where line 27 does not exist), COMPSEE BASIC uses the next higher line number (see Example 2). If the non-existent line number is the end of a LIST range (e.g., LIST 10-55 where line 55 does not exist), COMPSEE BASIC uses the next lower line number (see Example 3).

2. A LIST of a RAM file first sends a header record out the RS-232-C port followed by a carriage return linefeed. The header record is in the form: reclen,recnt,B where reclen is the file record length and recnt is the record count. The B indicates that the file is listed in binary form. Reference Section 24 - COMMUNICATIONS for more information on data modes (Data Bits and Parity).
3. A LIST can be terminated by pressing <ALT> <C> from the keypad. The <ALT> <C> forces an error 24 and returns control to IMMEDIATE mode.
4. If the file contains characters with ASCII values greater than 127, use eight data bits, no parity mode.

EXAMPLES:

Example 1 - This lists the file FILEA out the RS-232-C port. For Examples 2 and 3, assume that the following program is stored in RAM:

```
LIST     "FILEA"

10 FOR I = 1 TO 5
20 A = A + I
30 NEXT I
40 PRINT A
```

Example 2 - This sends lines 20 and 30 out the RS-232-C port.

```
LIST 11-30
```

Example 3 - This sends lines 20, 30, and 40 out the RS-232-C port.

```
LIST 20-55
```

***LOAD***

DEFINITION: Receive a RAM file over the RS-232-C port and store it.

FORMAT: LOAD "filename"

filename - A character string which represents a valid file name.

NOTES:

1. The RAM file data must be preceded by a header record in the form:

   rlen,rcount[,dtype]crlf

   - rlen is the file record length; the valid range is between 1 and 2048, inclusive.
   - rcount is the number of records to transmit; 0 to 65535, inclusive (0 creates an empty file).
   - dtype is the mode in which the data is transmitted; B for binary and H for HEX ASCII. If the dtype is not included, the default is binary.crlf indicates that the header must be followed by a carriage return and a linefeed.

2. The data must follow immediately after the header record.
3. A new file can be loaded over an existing file; however, the existing file must first be closed. If the new file is larger than the old file, the old file is deleted and the new file added. If the new file is smaller or the same size as the old file, the new file is written over the old file. But, the leftover memory is still allocated to that file. For example, file TEST is allocated 8192 bytes. A new TEST file is loaded which needs 2048 bytes. The leftover 6144 bytes are still allocated to the TEST file. In order to free the 6144 bytes for use elsewhere, the TEST file must first be deleted (using the KILL command) and then reloaded.
4. Do not use XON/XOFF flow control if the file being loaded contains XON (ASCII value 17) or XOFF (ASCII value 19) characters as data characters. The Apex II device will interpret these data characters as flow control characters. Also, if the file contains characters with ASCII values greater than 127, use eight data bits, no parity mode.
5. A special version of the LOAD command activates the operating system bootstrap loader routine resident in FLASH:

   LOAD "!!!!!!!!!H"

   The loader routine interprets all data received through the RS-232-C port as part of a new COMPSEE BASIC operating system which must be an Intel HEX formatted file

> ***WARNING***
>
> The data sent to the loader routine must be from the update file provided by Compsee Inc.

Once the new operating system is loaded, the Apex II terminal reboots, thereby deleting any program, data, or files. The RS-232-C port is also reset to the default values (see MODE statement). However, the RS-232-C port is not reset if the following command is issued:

EXAMPLE:

```
LOAD     "TESTER"
```

The file TESTER is loaded into memory. The first record must be the header record.

An example of a header record is: `100,3,B`
This loads three 100 byte records in binary form.

## ***LOC Function***

DEFINITION: Return the number of characters in a buffer or a RAM file.

FORMAT: x = LOC (fnum)

fnum -An arithmetic expression with the following values:

| | | |
|---|---|---|
| 1 | = | Keypad buffer |
| 4 | = | RS-232-C buffer |
| 5 - 255 | = | RAM file |

NOTES:

1. If the buffer or file is empty, a 0 is returned.
2. If FNUM refers to a RAM file number which is not associated with an opened file, a -1 is returned.
3. The maximum number of characters the keypad buffer can hold is 31; the maximum number for the RS-232-C buffer is 254.
4. The LOC function is the complement of the LOF Function.

EXAMPLE:

```
10 OPEN "TEST" AS #10 LEN = 20
20 WRITE #10, USING "C5", "12345"
30 PRINT LOC(10),LOC(5)
RUN
20   -1
```

File 10 has 1 record with 20 characters; the total number of characters in the file is 20. File number 5 is not associated with a file, so a -1 is returned.

## ***LOCATE***

DEFINITION: Position the cursor on the screen.

FORMAT: LOCATE row,column

row - The row to put the cursor in

column - The column to put the cursor in

NOTES:

1. LOCATE 4,17 turns the cursor off
2. LOCATE 1,17 generates an ERROR 14

EXAMPLES:

Example 1 - Move the cursor to the second row, first column.

```
LOCATE   2,1
```

Example 2 - Turn the cursor off.

```
LOCATE    4,17
```

## LOCK

DEFINITION: Disable the LIST command.

FORMAT: LOCK "code"

code - A character expression which represents the lock code. Valid length: 1 to 40.

NOTES:

1. Once an Apex II unit is locked, the LIST command is disabled until the unit is unlocked (see the UNLOCK statement).

EXAMPLE:

```
LOCK     "123456"
```

The Apex II terminal is locked using the code "123456".
The LIST command is disabled until the unit is unlocked.

## LOF Function

DEFINITION: Return the number of characters that will fit in the RS-232-C port or the keypad buffer.

FORMAT: x = LOF (fnum)

fnum -An a.e. with the following values:

1 = Keypad buffer
4 = RS-232-C buffer

NOTES:

1. The value returned is the maximum number of characters allowed in the buffer minus the number of characters in the buffer. The maximum number of characters for the keypad buffer is 31; the RS-232-C maximum is 254.
2. The LOF function is the complement of the LOC function.

EXAMPLE:

```
PRINT    LOF(1)
```

The sends the number of characters that will fit in the keypad buffer out the RS-232-C port.

***MODE***

DEFINITION: Configure the RS-232-C port.

FORMAT: MODE baud, status, "type"

baud - The baud rate; valid numbers are:

0 - 300 baud
1 - 600 baud
2 - 1200 baud
3 - 2400 baud
4 - 4800 baud
5 - 9600 baud
6 - 19200 baud

status - The configuration of the RS-232-C port. The following numbers are added together to form the status byte:

1 - XON/XOFF flow control (the default is: XON/XOFF).

2 - 7 data bits, parity enabled (the default is: 7 data bits, parity enabled)

4 - Odd parity (the default is: even parity)

8 - 2 stop bits (the default is: 1 stop bit)

The absence of a number causes COMPSEE BASIC to default to the description in parentheses. If the default of 8 data bits is selected, COMPSEE BASIC ignores the parity selection. The only data/parity choices are:

- 8 data bits, no parity
- 7 data bits, even parity
- 7 data bits, odd parity

The following chart defines all of the possible options for flow control and status bits:

| Status | Flow | Data Bits | Parity | Stop |
|---|---|---|---|---|
| 0 | RTS/CTS | 8 | N | 1 |
| 1 | XON/XO | 8 | N | 1 |
| 2 | RTS/CTS | 7 | E | 1 |
| 3 | XON/XO | 7 | E | 1 |
| 4 | RTS | 8 | O | 1 |
| 5 | XON/XO | 8 | O | 1 |
| 6 | RTS/CTS | 7 | O | 1 |
| 7 | XON/XO | 7 | O | 1 |
| 8 | RTS/CTS | 8 | N | 2 |
| 9 | XON/XO | 8 | N | 2 |

| Status | Flow | Data Bits | Parity | Stop |
|---|---|---|---|---|
| 10 | RTS/CTS | 7 | E | 2 |
| 11 | XON/XO | 7 | E | 2 |
| 12 | RTS/CTS | 8 | O | 2 |
| 13 | XON/XO | 8 | O | 2 |
| 14 | RTS/CTS | 7 | O | 2 |
| 15 | XON/XO | 7 | O | 2 |

EXAMPLES:

*A status of 7 means:*
1 = XON/XOFF flow control
2 = 7 data bits
4 = Odd parity
= 1 stop bit
(Status does not include the number 1)

*A status of 2 means:*
= RTS/CTS flow control
(Status does not include the number 1)
2 = 7 data bits
= Even parity
(Status does not include the number 4)
= 1 stop bit
(Status does not include the number 8)

*A status of 15 means:*
1 = XON/XOFF flow control
2 = 7 data bits
4 = Odd parity
8 = 2 stop bits

"type"- The data mode; B for binary mode and H for HEX/ASCII.
If the type letter is omitted, binary mode is used.

NOTES:

1. COMPSEE BASIC always uses 1 start bit.
2. The Apex II terminal is capable of full-duplex (i.e., it can transmit and receive data at the same time) and half-duplex RS-232-C operation. The Apex II terminal does not echo the RS-232-C characters it receives.
3. The present RS-232-C settings can be determined by using the PEEK function and performing several calculations. RS-232-C communications supports odd parity and two stop bits. The PEEK memory address is 22252. Therefore, an application program which uses address 22203 to calculate the present communications settings of the Apex II terminal does not change. However, if any of the added communications features are used (i.e., 2 stop bits or odd parity), address 22252 and the new parsing calculations must be used. See Example 2 for the parsing calculations.
4. After initial power up, the Apex II terminals is set to 9600 baud, 7 data bits, even parity, RTS/CTS flow control, and binary data mode.
5. See Section 24 for more information on data communication via the RS-232-C port.

EXAMPLES:

Example 1

```
MODE 2,1
```

This sets the RS-232-C port to 1200 baud, binary mode, 8 data bits, no parity, 1 stop bit, and XON/XOFF flow control.

Example 2 The following programs are examples of how to determine the present communications setting of an Apex II terminal. The BAUD variable corresponds to a value which can be used as the baud input parameter to the MODE statement; the STATUS variable corresponds to a value which can be used as the status input parameter to the MODE statement.

```
10 S = PEEK (22252) 'Fetch the setting
20 BAUD = S AND 7 ' Calc the baud
30 'DTYPE is the data type
40 '0 = binary 1 = HEX/ASCII
50 DTYPE = (S AND 8)/8
55 'Calculate the status configuration
60 STATUS = (S AND 240)/16
70 'Change to binary data mode, 2 stop bits
80 NEWSTAT = STATUS OR 8 'Set for 2 stop bits
90 MODE BAUD,NEWSTAT,"B"
```

## ***NEW***

DEFINITION: Delete the application program and clear all variables and arrays.

FORMAT: NEW

NOTES:

1. NEW closes all files and turns off all DIAGNOSTIC mode options.
2. A NEW processed from within a program forces COMPSEE BASIC back into IMMEDIATE mode.

EXAMPLE:

```
NEW
```

Delete the application program.

### ***NOT UNARY Function***

DEFINITION: Return the one s complement of an arithmetic expression.

FORMAT: X = NOT(num)

num - An arithmetic expression from 0 to 65535, inclusive.

NOTES:

1. A negative operand causes an error
2. The NOT function returns a one's complement without regard to a sign bit; NOT (1) is 65534 and NOT (1000) is 64535. The NOT operator in some BASICs returns the two's complement of the number, using the most significant bit as the sign bit. In those BASICs, NOT 1 is -2 and NOT 1000 is -10001.

EXAMPLES:

Example 1 - Sends 65280 to the RS-232-C port.

```
10 A = 255
20 PRINT NOT(A)
```

Example 2 - The variables X and Y are compared on line 30; A is set to false because X equals Y. The condition A is checked on line 40. Since A is false, NOT(A) is true (i.e. NOT(A) is a number other than 0) and "NOT EQUAL" is printed to the RS-232-C port.

```
10 X = 2
20 Y = 2
30 A = (X<>Y)
40 IF NOT (A) THEN PRINT "NOT EQUAL"
```

### ***ONBAR***

DEFINITION: Enable the COMPSEE BASIC bar code interrupt.

FORMAT: ONBAR GOSUB line number
or
ONBAR GOSUB label

line number- application program line which is the first line of the subroutine.

label - application program label which is the first line of the subroutine. Refer to the LABEL section of this part of the manual for further information on labels.

NOTES:

1. The ONBAR statement allows COMPSEE BASIC to interrupt itself and retrieve a bar code. If ONBAR is invoked, the bar code buffer is checked for bar codes at the beginning of every COMPSEE BASIC statement. If bar codes are present, the subroutine specified in the ONBAR statement is executed.
2. While an ONBAR routine executes, the COMPSEE BASIC communications interrupt is automatically disabled. However, the hardware interrupt is not affected, so no bar codes are lost. If, upon exiting the ONBAR subroutine, bar codes remain in the receive buffer, the ONBAR subroutine re-executes automatically.
3. A line number value of 0 (ONBAR GOSUB 0) disables the COMPSEE BASIC barcode interrupt.

EXAMPLE:

```
10  ONBAR GOSUB GETBAR
20  TMP$=TIME$
30  IF TMP$<>TIME$ THEN PRINT #0, USING "P1,C10", TIME$
40  GOTO 30
100 GETBAR: A$ = BARCODES
110 IF A$ = "Z" THEN PRINT #0 USING "P17,C15","BAD READ": ELSE PRINT #0, USING"P17,C15",A$(2:16);
120 RETURN
RUN
```

The screen is continuously updated with the time. If COMPSEE BASIC detects a character in the bar code buffer, program control is passed to the GETBAR: routine. After the screen is updated with the bar code value, control is passed back to the time loop.

### ***ONCOM***

DEFINITION: Enable the COMPSEE BASIC communications interrupt.

FORMAT: ONCOM GOSUB line number
or
ONCOM GOSUB label

line number- application program line number which is the first line of the subroutine.

label - application program label which is the first line of the subroutine. Refer to the LABEL section of this manual for further information on labels.

NOTES:

1. The ONCOM statement allows COMPSEE BASIC to interrupt itself and retrieve data from the RS232-C receive buffer. If ONCOM is invoked, the receive buffer is checked for characters at the beginning of every COMPSEE BASIC statement. If characters are present, the subroutine specified in the ONCOM statement is executed.
2. While an ONCOM subroutine executes, the COMPSEE BASIC communications interrupt is automatically disabled. However, the hardware interrupt is not affected, so no bar codes are lost. If, upon exiting the ONCOM subroutine, bar codes remain in the receive buffer, the ONCOM subroutine re-executes automatically.
3. A value of 0 (ONCOM GOSUB 0) disables COMPSEE BASIC communications interrupt.

EXAMPLE:

```
10  ONCOM GOSUB COMDATA
20  TMP$ - TIME$
30  IF TMP$ <> TIME$ THEN PRINT #0, USING"P1,C10",TIME$;:TMP$ = TIME$
40  GOTO 30
100 COMDATA: INPUT$ #4, LOC(4), A$
110 PRINT #0, USING "P1,C16",A$
120 RETURN
RUN
```

The screen is continuously updated with the time. If COMPSEE BASIC detects a character in the receive buffer, program control is passed to the COMDATA routine. The data is read from the buffer and displayed on the screen. The LOC(4) function tells the INPUT$ statement to input as many characters as are in the receive buffer.

### ***ONERR***

DEFINITION: Enables error trapping. On an error, program executive jumps to a specified line.

FORMAT: ONERR GOTO line number
or
ONERR GOTO label

line number - application program line number.

label - application program label. Refer to the label section of this manual for further information on labels.

NOTES:

1. A line number value of 0 (ONERR GOTO 0) disables error trapping. Subsequent errors cause COMPSEE BASIC to enter the DIAGNOSTIC mode. An ONERR GOTO 0 statement within an error trapping routine causes COMPSEE BASIC to enter the DIAGNOSTIC mode immediately. The trapping is to execute an ONERR GOTO 0 for all errors which have no recovery action.
2. If an error occurs during the execution of an error trapping routine, COMPSEE BASIC immediately goes into the DIAGNOSTIC mode noting the second error; the first error is lost. No error trapping occurs within the error handling routine.
3. The RESUME statement must be used to exit from an error trapping routine; only a RESUME clears the error condition. See the RESUME statement in this manual.

EXAMPLES:

```
5  ONERR GOTO ERRHANDL
10 INPUT$ #4,5,T$
50 STOP
100ERRHANDL:
105IF ERR <> 19 THEN ONERR GOTO 0
110PRINT #0, USING "P1,C12", "NO RESPONSE";
120PRINT #0 USING "P17,C16", "1-Retry    2-Quit";
125DUM$ = INKEY$
130IF DUM$ = "1" THEN RESUME
135IF DUM$-"2" THEN RESUME NEXT
140GOTO 125
RUN
```

This example traps for a timeout on the RS-232-C port. The user is given the option to retry the INPUT or to end the program.

## ***OPEN***

DEFINITION: Open a RAM file

FORMAT: For new files:
OPEN filename AS [#] filenumber LEN = reclen

For existing files:
OPEN filename AS [#] filenumber

filename - A character string which represents a valid file name.

filnum - An arithmetic expression with a value between 5 and 255, inclusive.

reclen - An arithmetic expression with a value between 1 and 2047, inclusive.

NOTES:

1. An existing file cannot have a LEN = option.
2. The file number is associated with the file until the file is closed. All other I/O statements reference the file through the file number. Therefore, a RAM file must be OPENed before I/O operation can be done.
3. OPEN sets the record pointer to 0.
4. A file name of "***" is invalid. This is reserved for KILL all files option (see KILL statement).

EXAMPLES:

Example 1 - Opens a new file called NEWONE as file number 20; each record is 5 bytes long.

```
A$ = "NEWONE"
OPEN A$ AS #10*2 LEN = 20/4
```

Example 2 - ERROR in line 30 - file "ABC" already exists.

```
10 OPEN "ABC" AS #40 LEN = 30
20 CLOSE #40
30 OPEN "ABC" AS #50 LEN = 30
```

## ***OUT***

FORMAT: OUT parameter

parameter - number between 0 and 63, inclusive.

NOTES:

1. OUT uses the following numbers to set the system flags:

```
  1     =   RTS
  2     =   STOP_MOTOR
  4     =   NO_SCAN_BIT
  8     =   DST
 16     =   NO_RELOAD
 32     =   NO_ALT_C
 64     =   (Unused)
128  =   (Unused)
```

The presence of the number in the parameter value sets the flag. The absence of the number clears the flag. For example, a parameter value of 10 sets the DST & STOP_MOTOR flags and clears the RTS, NO_SCAN_BIT, NO_ALT_C, and NO RELOAD flags. A parameter value of 7 sets the RTS, STOP_MOTOR< & NO SCAN BIT flags and clears the DST, NO_ALT_C, and NO_RELOAD flags.

| FLAG | IF SET | IF CLEAR |
|---|---|---|
| RTS | RTS RS-232-C control line is not asserted. | RTS RS-232-C control line is asserted. |
| STOP_MOTOR | The mirror will not reciprocate. | The mirror will reciprocate during the scan. |
| NO_SCAN_BIT | Scanning is disabled. | Scanning is enabled. |
| DST | Automatic clock compensation for daylight savings time is enabled. | Automatic clock compensation for daylight savings time is disabled. |
| NO_RELOAD | Power down timer is not reloaded at the beginning of every COMPSEE BASIC statement. | Power down timer is reloaded at the beginning of every COMPSEE BASIC statement. |
| NO_ALT_C | The INPUT$ #4 statement accepts a ALT / C as a character. | The INPUT$ #4 statement aborts upon receiving a ALT / C character. |

2. Values 64 and 128 are ignored. With a parameter value of 143, COMPSEE BASIC sets the RTS, STOP_MOTOR, NO_SCAN_BIT, & DST flags (a parameter value of 15) and ignores the 128. With a parameter value of 65, COMPSEE BASIC sets the RTS flag (parameter value of 1) and ignores the 64.

3. A parameter value greater than 255 causes an error.
4. The NO_SCAN_BIT and DST flags can also be set and cleared by the SYSPARMS statement.
5. The STOP_MOTOR, NO_SCAN_BIT, NO_RELOAD, NO_ALT_C, and DST flags are all clear at initial power up.
6. The flags can be read using the INP system variable.
7. Once data is entered through any of the following means, the NO_RELOAD flag is cleared and the power down timer is reset before every COMPSEE BASIC statement is executed.

   - Using the RS-232-C port
   - Pressing a key on the Apex II terminal keypad
   - Scanning a bar code

EXAMPLES:

Example 1 - Set the RTS and DST flags, clears all other flags.

```
OUT 9
```

Example 2 - Sets only the no reload flag and leaves all other flags alone.

```
X = INP
OUT (16 OR X)
```

Example 3 - To clear just the no reload flag, you would use:

```
X = INP
OUT (239 AND X) '239 is NOT 16 or 255-16
```

## ***PAUSE***

DEFINITION: Put the Apex II into a low power mode for a specified amount of time.

FORMAT: PAUSE x

x - An arithmetic expression which represents the idle timeout. Valid values are 0 to 255.

NOTES:

1. The Apex II will return from low power mode after x seconds.
2. If issued from immediate mode, the Apex II will continue execution with the next program statement.

EXAMPLES:

Example 1 - Puts the Apex II into low power mode and sets the timeout to 10 seconds.

```
PAUSE 10
```

Example 2 - Puts the Apex II into low power mode until a key is hit.

```
PAUSE 0
```

### ***PEEK Function***

DEFINITION: Return the value of a byte in memory.

FORMAT: x= PEEK (addr)

addr - Address of the byte to be read. The valid range is 0 to the top of memory. The top of memory address depends on the amount of memory in the Apex II terminal.

NOTE:

```
The return byte has an integer value between 0 and 255, inclusive.
```

EXAMPLE:

```
PRINT PEEK(50000)
```

This sends the value of the byte located at memory address 50000 out the RS-232-C port.

### ***POKE***

DEFINITION: Put a byte into memory.

FORMAT: POKE address, data

address - Address in memory where the data is to be written. The valid address range is from 0 to the top of memory; the top of memory depends on the amount of memory in the Apex II terminal.

data - The data to be written to memory; the valid range is 0 to 255, inclusive.

NOTES:

EXAMPLE:

```
POKE 65000,250
```

This command writes the value 250 to memory address 65000.

### ***PRINT***

DEFINITION: The PRINT statement has three definitions:

1. Formatted print to the RS-232-C port
2. Formatted print to the screen
3. Unformatted print to the RS-232-C port

FORMAT: *Formatted:* PRINT #filenum, USING format,list[;]
or
*Unformatted:* PRINT list [;]

filenum - An a.e. with a value 0 or 4
0 = screen
4 = RS-232-C buffer

format - An f.s. (see FORM for a detailed explanation of FORM strings).

list - A list of character or arithmetic expressions, or variables, separated by commas.

NOTES:

1. A semi-colon suppresses writing a carriage return and a linefeed (CRLF) at the end of the print.
2. PRINT #4, PRINT #0 and PRINT write a CRLF. If the screen cursor is in the second row, a PRINT #0 moves the cursor off the screen.
3. The screen scrolls as data is written to it. If the PRINT starts with the cursor in row 1 and the data does not fit in row 1, the rest of the data is printed in row 2.
4. An unformatted PRINT prints five 16 character fields per line. After 80 characters, a CRLF is written. If data overflows its field, it is continued in the next field. The next piece of data starts in the N + 1 field. If data is less than 16 characters, the rest of the field is padded with blanks.

EXAMPLES:

Example 1 - Sends "ABEL" to the RS-232-C port.

```
PRINT #4, USING "C4", "ABEL"
```

Example 2 - Prints "JOE" to the screen starting in row 1, column 1 and CRLF is suppressed.

```
PRINT #0, USING "P1,C3","JOE";
```

Example 3 - Sends the number 9 and the string "CAIN" to the RS-232-C port.

```
PRINT 9, "CAIN"
```

Example 4 - Writes a CRLF to the RS-232-C port.

```
PRINT
```

## ***READ***

DEFINITION: The READ statement has two definitions:
1. Read a formatted record, from a RAM file
2. Read a data statement

FORMAT: *Formatted:* READ #filenum, USING format, [rec = recnum],list
or
Read data: READ list

filenum - An arithmetic expression with a value between 5 and 255, inclusive.

format - A form string (see FORM for a detailed explanation of FORM strings).

recnum - Record number of record to read.

list - A list of variables or array elements, separated by commas, to be read from the file or the DATA statement.

NOTES:

*For data read:*

1. A READ immediately followed by a list defaults to a read of a DATA statement. DATA elements are read in sequential order. Each subsequent READ updates the DATA element pointer. A RESTORE command resets the DATA element pointer (see RESTORE for further details).

*For formatted read:*

1. A file must be opened before it can be read.
2. If the REC = is omitted, the record pointer is incremented by one and the next record is read. If the REC = option is used, the record number must be between 1 and the record count, inclusive.
3. The record pointer is updated to point to the record just read.

*For both reads:*

1. Data can be read directly into an array element or a substring.

EXAMPLES:

Example 1 - Assumes file number 10 is associated with an open file: read the first 3 bytes of record 2 in file 10 and put them into the variable A$; read the next 3 bytes and put them into array element B$(4); and read the next 3 bytes and put them into string C$ starting at position 1. Read the section on substrings in this manual before READing data directly into a substring.

```
DIM B$(5)
READ #10, USING "C3,C3,C3",
REC = 2,A$,B$(4),C$(1:3)
```

Example 2 - Reads the DATA statement and puts "1" into A$, "2" into B$, and "3" into C$.

```
10 DATA "1","2","3"
20 READ A$,B$,C$
RUN
```

## ***REMARK Lines***

DEFINITION: Add an explanatory comment to the application program.

FORMAT: 'remark

remark - any ASCII characters

NOTES:

1. Remarks are not executed; however, they do take up space in memory and slightly slow down execution time.
2. Remarks are stored exactly as they are entered.
3. Because COMPSEE BASIC considers everything after the ' and up to the carriage return linefeed part of the remark.
4. The remark must always be the last statement on the line.

EXAMPLE:

```
10  A = 5 : B = 7 'This is a remark at the end of  a line
20  'THIS IS A REMARK ON A SEPARATE LINE
```

### ***RESTORE***

DEFINITION: Reset the DATA element pointer so that DATA statements can be reread.

FORMAT: RESTORE [line number or label]

line number - application program line number

label - application program label. See the LABEL section of this manual for further information on labels.

NOTES:

1. If the line number or label is not present, the pointer is reset to the first DATA statement; i.e., the next READ statement will access the first DATA element.
2. If the line number or label is specified, the next READ will access the DATA statement at that line number or label. All subsequent READs will access the DATA elements sequentially from that point.
3. If the line number or label does not point to a DATA statement, an error results.

EXAMPLE:

```
10 DATA 1,2,3
20 READ A,B,C
30 RESTORE
40 READ D,E,F
RUN
```

A and D contain 1; B and E contain 2; C and F contain 3.

### ***RESUME***

DEFINITION: Return from an error trapping routine to normal program execution.

FORMAT: RESUME [line number or label]
or
RESUME NEXT

line number - application program line number.

label - application program label. Refer to the LABEL section of this manual for further information on labels.

NOTES:

1. A RESUME without a label or a line number re-executes the statement which caused the error. A RESUME NEXT executes the statement immediately following the one that caused the error. A RESUME followed by a line number or label resumes execution at the specified line.
2. A RESUME statement not within an error trapping routine causes an error.

EXAMPLE:

```
10  ONERR GOTO ERRHANDL
      .
      .
      .
100 ERRHANDL: IF ERR = 46 AND ERL = 80 THEN RESUME RETRY
```

If error 46 occurs on line 80, resume execution at label RETRY.

***Note***

When error trapping occurs on error 24, use RESUME, not RESUME NEXT.

## ***REWRITE***

DEFINITION: Rewrite a formatted record in a RAM file.

FORMAT: REWRITE #filnum, USING format, [REC = recnum], list

filenum - An arithmetic expression with a value between 5 and 255, inclusive

format - A form string (see FORM for detailed explanation of FORM strings).

recnum - Record number of record to REWRITE.

list - A list of variables, character expressions, or arithmetic expressions, separated by commas.

NOTES:

1. A file must be opened before it can be rewritten.
2. If the REC = is omitted, the record pointed to by the record pointer is rewritten. If the REC = option is used, the record number must be between 1 and the record count, inclusive.
3. A record rewrite is buffered. If an error occurs anywhere within a REWRITE statement, the record is not rewritten.
4. The record count is unchanged after a REWRITE.

EXAMPLES:

Example 1

REWRITE #10, USING "C3", "JOE"

Assuming file number 10 is associated with an open file, JOE is written to the record pointed to by the record printer.

Example 2

REWRITE #10, USING "N5.1", REC = 1.9

The first five bytes of record 1 in file 10 are rewritten with 2 spaces, a nine, a decimal point, and a 0 (refer to the FORM FIELD description for further details).

### ***RNUM Function***

DEFINITION: Return the number of the last record processed in a RAM file.

FORMAT: x = RNUM(fnum)

fnum -An a.e. with a value between 5 to 255, inclusive.

NOTES:

1. A -1 is returned if fnum is not associated with an opened file. A 0 is returned if no records have been processed.

EXAMPLE:

```
PRINT RNUM(10)
```

This sends the number of the last record processed in file 10 out the RS-232-C port. If file 10 is not opened, a -1 is sent.

### ***RTRIM Function***

DEFINITION: Removes all trailing occurrences of a specified character in the string, and returns a character string.

FORMAT: A$=RTRIM(A$,B$)

A$ - The return value.
B$ - The string to trim, Returned trimmed up.
C$ - The trim character.

EXAMPLES:

Example 1 -

```
A$=RTRIM(A$,B$)
```

Sets A$ =“XXXXXXX” if
A$ = “XXXXXXX_____” and B$ = “_”

Example 2 -

```
C$=RTRIM(A$,B$)
```

Sets C$ =“XXXXXXX” and does not change A$
if A$ = “XXXXXXX_____” and B$ = “_”

Example 3 -

```
C$=RTRIM(A$,B$)
```

Sets C$ =“X_X_X” and does not change A$
if A$ = “X_X_X_____” and B$ = “_”

### ***RUN***

DEFINITION: Execute the application program.

FORMAT: RUN

NOTES:

1. The RUN command may be issued either from IMMEDIATE mode or from within an application program. A RUN command from within an application program has been the effect of rerunning the program.
2. RUN closes all files, resets all variables and arrays, and turns off all DIAGNOSTIC mode options.

EXAMPLE:

```
10 FOR I = 1 TO 3
20 PRINT I
30 NEXT I
RUN
1
2
3
```

RUN executes the sample program.

### ***SAVE***

DEFINITION: Save the application program in the FLASH.

FORMAT: SAVE

NOTES:

1. The program is moved from RAM into FLASH and makes more RAM available for variable and file storage.
2. A NEW command will erase the program from FLASH.
3. The SAVE command will return an error if there is already a program saved in the FLASH.
4. The SAVE command will return an error if there is no program in the unit to save.

EXAMPLE:

```
SAVE
```

This moves the program from RAM into FLASH.

### ***SGN Function***

DEFINITION: Return the sign of an arithmetic expression.

FORMAT: X = SGN(num)

num - An arithmetic expression with a valid value as described in the number section of this manual.

NOTES:

1. A positive number returns a one, a negative number returns a minus one, and a zero returns a zero.

EXAMPLE:

```
10 A = 12
20 B = -2000
30 C = 0
40 PRINT SGN(A), SGN(B), SGN(C)
```

This sends 1, -1, and 0 to the RS-232-C port.

### ***SIGNAL***

DEFINITION: Specify the operation of the LED and the beeper.

FORMAT: SIGNAL number

number - An arithmetic expression with a value between 0 to 4, inclusive.

- 0 - Enables good read indication on a valid bar code scan.
- 1 - Disables good read indication on a valid bar code scan.
- 2 - Good read indication; light the LED and sound the beeper for ½ second continuously.
- 3 - Flash the LED and turn the beeper on and off in rapid succession.
- 4 - Flash the LED and turn the beeper on and off at a slower rate than SIGNAL 3.
- 5 - Enables key click for key depression.
- 6 - Disables key click for key depression.

NOTES:

1. A SIGNAL 2, SIGNAL 3, or SIGNAL 4 immediately following a BEEP overrides the BEEP.

EXAMPLE:

```
10 FOR I = 1 TO 5
20 IF I = 3 THEN SIGNAL 3
30 NEXT I
RUN
```

This example flashes the LED and turns the beeper on and off when the loop variable is 3.

### ***SIN Function***

| |
|---|
| This function is not supported. |

*SOFTSCAN Function*

DEFINITION: Software trigger for the integrated laser on the Apex II.

FORMAT: SOFTSCAN

### ***SQR Function***

| |
|---|
| This function is not supported. |

### ***STOP***

DEFINITION: Halt execution of the application program and enter DIAGNOSTIC mode.

FORMAT: STOP

NOTES:

1. When COMPSEE BASIC executes a STOP statement, the following message is displayed on the first line of the screen:

   ```
   BREAK line
   ```

   where line is the line number of the STOP statement. The first position of the second line displays the prompt >, indicating the Apex II terminal is ready to accept DIAGNOSTIC mode commands.
2. The STOP statement can only be used in RUN mode.

EXAMPLE:

```
10 FOR I = 1 TO 5
20 IF I = 4 THEN STOP
30 PRINT I : NEXT
RUN
1
2
3
```

When I equals 4, execution of the program halts and the Apex II terminal enters DIAGNOSTIC mode. If DIAGNOSTIC mode is terminated, a CONT command can be entered to resume execution. See the CONT command description in this manual.

## ***STR$ Function***

DEFINITION: Return a string representation of the value of a numeric expression.

FORMAT: x$ = STR$(num)

num - An arithmetic expression with a valid value as described in the Section 11 - NUMBERS of this part of the manual.

NOTES:

1. A leading "+" causes an error (e.g., STR$( + 1224)). COMPSEE BASIC uses a blank to signify a positive number.
2. The maximum number of characters possible is 16 (e.g., -1.2345678 E -120).
3. The sign indicator of a positive number is not returned as a character. In some BASICs, STR$(123) returns four characters: a blank for the sign and 123. COMPSEE BASIC Interpreter does not return the leading blank.

EXAMPLES:

Example 1

```
10 A$ = STR$(1234)
20 PRINT A$
RUN
```

This sends a "1234" to the RS-232-C port.

Example 2

```
10 A$ = "1234"
20 B$ = STR$(5678)
30 PRINT #0s, USING, "P1,C8", A$ + B$
RUN
```

This prints "12345678" to the screen starting at row 1, column 1.

## ***STRING$ Function***

DEFINITION: Return a string of a specified length whose characters all have the same ASCII codes.

FORMAT: X$ = STRING$(length, code)

length- An arithmetic expression with a value between 0 to 255, inclusive, representing the length of the string.

code- An arithmetic expression with a value between 0 and 255, inclusive, representing the ASCII code for the characters in the string.

NOTES:

1. If the RS-232-C port is set for 7 data bits, sending a STRING$ with an ASCII code between 128 to 255 causes a parity error. See the MODE statement in this manual for more information on setting the RS-232-C port.
2. If the STRING$ is assigned to a variable and the STRING$ length is greater than the default limit, the variable must first be dimensioned.

EXAMPLE:

```
PRINT STRING$(10,65)
```

This sends a string of ten "A" to the RS-232-C port.

## ***SYSPARMS***

DEFINITION: Specify certain Apex II terminal parameters.

FORMAT:

SYSPARMS noscan,pdt,dto,decode1,decode2,decode3,2of5,coda,dst,Plessey,contrast

***NOTE***

The proper format for the SYSPARMS statement is as shown above, where all the parameters appear on one line. The SYSPARMS statement will cause an error if it is keyed into your program on multiple lines.

**noscan - NO SCAN OPTION**

The scanner or wand is disabled with a value of 1, enabled with a value of 0. At initial power up, this value is 0.

**pdt - POWER DOWN TIMER**

The amount of time, in seconds, that COMPSEE BASIC waits in a no "activity" state before the unit transitions to a sleep mode.

An "activity" is defined as: Pressing a key on the keypad; executing a COMPSEE BASIC statement; using the RS-232 port.

Any activity resets the timer. The valid range of timer values is 0 to 255 x 7.5 seconds. A value of 0 or 1 means the unit will never power down. The value at initial power up is 90 seconds. The current value of pdt is located at address 284.

**dto - DEVICE TIMEOUT**

The amount of time, in seconds, that COMPSEE BASIC will wait to print a character out or input a character from the RS-232-C port.

If COMPSEE BASIC is attempting to input a character from the RS-232-C port, COMPSEE BASIC will wait DEVICE TIMEOUT seconds for the character before timing out. If COMPSEE BASIC is attempting to print a character out the RS-232-C port, COMPSEE BASIC will try for DEVICE TIMEOUT seconds before timing out.

**decode1, decode2, decode3 - DECODER CONTROL PARAMETERS**

The types of bar code the Apex II terminal will decode can be set by the operator. Each bar code type has an assigned value. The total of the values tells COMPSEE BASIC which bar codes to decode.

decode1

| | |
|---|---|
| 1 = | decode Code 3 of 9 |
| 2 = | decode Interleaved 2 of 5 |
| 4 = | decode Codabar |
| 8 = | decode Code 128 |
| 16 = | decode Code 93 |
| 32 = | decode Code 11 |
| 64 = | decode UPC-A |
| 128 = | decode UPC-E |

A decode1 value of 48 tells COMPSEE BASIC to decode Code 93 and Code 11 bar codes only. A decode 1 value of 255 tells COMPSEE BASIC to decode all 8 of the code types listed. The value at initial power up for decode1 is 255. The current value of decode1 is located at address 276. The decode2 and decode 3 values includes other decoder parameters for the operator to choose.

decode2

| | |
|---|---|
| 1 = | decode EAN-8 |
| 2 = | decode EAN-13 |
| 4 = | any Codabar symbol that begins or ends with a "D" must be part of a pair |
| 8 = | all UPC and EAN symbols must have a 2 or 5 character supplemental |
| 16 = | do a redundancy check; check for 2 identical scans |
| 32 = | Code 11 has 1 check digit instead of 2 |
| 64 = | MSI/Plessey has 2 check digits instead of 1 |
| 128 = | Not Used |

The decode2 default value is 3. The current value of decode2 is located at address 277.

decode3

| | |
|---|---|
| 1 = | decode Industrial 2 of 5 |
| 2 = | decode Matrix 2 of 5 |
| 4 = | decode Identicode 2 of 5 |
| 8 = | decode MSI/Plessey |
| 16 = | decode 2 of 5 Symbology with Required Check Digit |
| 32 = | decode Code 3 of 9 full ASCII |
| 64 = | Not Used |
| 128 = | Not Used |

The decode3 default value is 0. The current value of decode3 is located at address 285.

2of5 INTERLEAVED 2 OF 5 CHARACTERS

The number of characters in the Interleaved 2 of 5 bar codes the operator will be scanning. At power up, the value of 2of5 is 14. The current value of 2of5 is located at address 274. COMPSEE BASIC accepts values from 1 to 255.

coda CODABAR CHARACTERS
The minimum number of characters in the Codabar bar codes the operator will be scanning. This number includes the start and stop characters. The value at initial power up is 5. The current value of coda is located at address 275. COMPSEE BASIC accepts values from 1 to 255.

dst DAYLIGHT SAVINGS TIME
A value of 1 tells COMPSEE BASIC to automatically set the system clock to daylight savings time the last Sunday in April and to reset the system clock to standard time the last Sunday in October. A 0 value disables this adjustment. The value at initial power up is 0.

Plessey MSI PLESSEY CHARACTERS
The will represent the minimum number of characters in the MSI Plessey that the operator will be scanning. This number includes the 1 or 2 check characters. The value at initial power up is 0. COMPSEE BASIC accepts values from 1 to 255.

NOTES:

1. The values of the SYSPARMS statement must be put in the exact order shown above. If the operator does not want to change a certain parameter, that parameter's position in the list must be marked by a comma.
2. The SYSPARMS values retain their current value until explicitly changed. For example, on power up the dto value is 5 seconds. The value is then changed to 20 seconds by the following statement:

```
SYSPARMS ,,20,,,,,,,,
   If the next SYSPARMS statement changed pdt to 60 seconds:
SYSPARMS ,8,,,,,,,,,
   The dto value is still 20 seconds.
```

EXAMPLES:

Example 1 - Turns off the power down timer and leaves the rest of the parameters unchanged.
```
SYSPARMS ,0,,,,,,,,,
```

Example 2 - This decodes UPC-E bar codes only. It leaves the rest of the parameters unchanged.
```
SYSPARMS ,,,128,0,0,,,,,
```

Example 3 - Sets the minimum number of MSI Plessey characters that will be scanned to 10 digits.
```
SYSPARMS ,,,,,,,,10,,
```

## ***TAN Function***

| |
|---|
| This function is not supported. |

## ***TIME$ System Variable***

DEFINITION: Set or retrieve the time.

FORMAT: *To set the time:* TIME$ = "dstring"
or
To retrieve the time: X$ = TIME$

dstring - The time string in the form HH:MM:SS, where:

HH- hour (from 0 to 23)

MM - minute

SS - second

NOTES:

1. The time is represented in twenty-four hour military format. For example, 1 p.m. is represented as: 13:00:00
2. COMPSEE BASIC always represents the TIME$ string in eight characters. Therefore, the hour, minute, and second are padded with zeros when necessary. For example, ten past 1 p.m. is represented 13:10:00

   Conversely, the leading zeros are not necessary when setting the time. For example 13:10:0 is sufficient to set the time to ten past 1 p.m.
3. Any data after the seconds is ignored by COMPSEE BASIC (see example 3).

EXAMPLES:

Example 1 - Stores the present time in variable TTIME$.

```
TTIME$ = TIME$
```

Example 2 - Sets the time to seven minutes and fourteen seconds past 5 p.m.

```
TIME$ = "17:7:14"
```

Example 3 - This sets the time to fifty-nine minutes and fifty-nine seconds past 11 p.m. The characters after the seconds are ignored.

```
TIME$ = "23:59:59 NEW DAY"
```

## ***UNLOCK***

DEFINITION: Enable the LIST command.

FORMAT: UNLOCK "code"

code - A character expression, with a length of one to forty, which represents the unlock code.

NOTES:

1. The UNLOCK code must be an exact match (in both content and length) of the LOCK code or the unit will not unlock. Refer to the LOCK statement in for details on locking an Apex II terminal.
2. If not unlocked after seven tries, the unit will stay locked; only a reboot will re-enable the LIST command.

EXAMPLE:

```
UNLOCK "123456"
```

The LIST command is re-enabled, assuming the unit was first locked using code "123456".

### ***VAL Function***

DEFINITION: Return the numerical value of a string.

FORMAT: x = VAL(string)

string - A character expression representing a numerical value. Characters 0 through 9, blanks, and leading plus and minus signs are valid.

NOTES:

1. Blanks are ignored and may be imbedded in the numeric string. For example, VAL("1 2 3 4") returns a value of: 1234.
2. A null string returns a 0.
3. The following characters are valid input for the VAL function: "0123456789+-" and a blank. All other characters can cause an error.
4. If "string" is out of the range of COMPSEE BASIC (see "Numbers"), VAL returns a 0.

EXAMPLE:

```
10 A$ = "-451"
20 A = VAL(A$)
30 PRINT A
RUN
```

This sends -451 to the RS-232-C port.

### ***VER$ System Variable***

DEFINITION: Return the COMPSEE BASIC version string.

FORMAT: x$ = VER$

NOTES:

1. The string returned by VER$ is always five characters long. If the string is less than five characters, the string is right-padded with blanks. For example, V1.1a is five characters; however, V1.2 is right-padded with a blank.

EXAMPLES:

```
PRINT VER$
V2.24
```

This sends the version to the RS-232-C port.

### ***WAND***

DEFINITION: Turn on the bar code wand.

FORMAT: WAND x

x - An arithmetic expression which represents the idle timeout. Valid values are 0 to 255, inclusive.

NOTES:

1. The bar code wand turns itself off after x seconds of inactivity. However, the time is refreshed with the x value every time a barcode is successfully decoded.
2. A WAND 0 statement turns off the bar code wand immediately.
4. If the WAND statement is issued on the Apex II unit without a bar code wand, an error 70 results.
5. The RS-232-C receive buffer is cleared when the wand is turned on.

EXAMPLE:

```
WAND 60
```

This turns on the wand and sets the idle timeout to 60 seconds.

### ***WHILE and WEND***

DEFINITION: Execute a series of statements as long as an expression is true.

FORMAT: WHILE test

.
.
.
WEND

test - any expression that resolves to a turn (not 0) or a false (0). Examples are:

WHILE 5
WHILE A = 6
WHILE B$<>"JOE"

Note that WHILE "test" is invalid because "test" does not evaluate to a true or a false. Also, WHILE 5 results in an infinite loop.

NOTES:

1. COMPSEE BASIC first evaluates the WHILE test. If the test is true, the statements up to the WEND are executed. When COMPSEE BASIC encounters the WEND, execution is looped back to the WHILE statement and the WHILE test is rechecked. If the test is still true, the loop is continued. If the test is false, execution continues at the statement after the WEND.
2. WHILE-WEND loops can be nested. Each loop must be terminated by a WEND. COMPSEE BASIC matches the WEND statement with the most recent WHILE statement (see example 2).

3. Loops are nested. This allows an unlimited number of loops within a program. The only restriction is that the loop nest cannot be greater than 11. A nest is a loop processed within another loop. For example:

```
 10WHILE I<10
 20WHILE J<3
 30WHILE K<5
 40PRINT I+J+K
 50K=K+1
 60WEND
 70K=1 : J=J+1
 80WEND
 90J=1 : I=I+1
100WEND
```

This program has a nest value of three.

A nest scheme of processing loops requires that all loops be resolved. This means that the loop counter must be greater than the loop end value. The first example must be changed to:

```
10 WHILE I <100
20 IF INKEY$ = "Q" THEN I = 1000 ELSE PRINT I
30 I=I+1
40 WEND
50 EXIT:
```

In this example, the loop counter (I) is set to the end value. The next time the WEND statement is processed, the loop terminates naturally.

EXAMPLES:

Example 1 - Sends the numbers 0, 1, 2, 3, and 4 to the RS-232-C port.

```
5  I=0
10 WHILE I <5 : I=I+1
20 PRINT I
30 WEND
RUN
```

Example 2 - Notice how the WEND matches with the most recent unmatched WHILE. The WEND in line 40 matches with the WHILE in line 20 and the WEND in line 50 matches the WHILE in line 10.

```
5  I=0
10 WHILE I<3: J=0
20 WHILE J <2 : J=J+1
30 PRINT I,J
40 WEND : I=I+1
50 WEND
RUN
0  1
0  2
1  1
1  2
2  1
2  2
```

## ***WRITE***

DEFINITION: Write a formatted record to a RAM file.

FORMAT: WRITE #filenum, USING format,list

filenum - An arithmetic expression with a value between 5 and 255, inclusive.

format - An form string (see FORM for detailed explanation of FORM strings).

list - A list of variables, character expressions, or arithmetic expressions separated by commas.

NOTES:

1. A file must be opened before anything can be written to it.
2. If the data written to the record is shorter than the record length, the record is padded with nulls.
3. A record write is buffered. If an error occurs anywhere within a WRITE statement, the record is not written to the file.
4. The record pointer and the record count are incremented before writing the buffer back to the file. The record pointer is always pointing to the end of the file after a WRITE.
5. The maximum number of records that can be written to a file is 65535.

EXAMPLE:

```
WRITE #10, USING "C3", "JOE"
```

Assuming file number 10 is associated with an open file, "JOE" is written to the end of the file.

# 17 ERROR MESSAGES

When COMPSEE BASIC detects an error in a program, program execution is halted and the Apex II terminal enters DIAGNOSTIC mode if the Apex II terminal is in BREAK mode 0 (see Section 23 for further details on BREAK modes). The information on the screen is saved and the error number and the error line are displayed on the top line of the screen. For further details, see the DIAGNOSTIC mode information in Section 6.

The DIAGNOSTIC mode can be disabled by using the ONERR statement. The operator can then write an error handler within the application program by using the ERR and ERL functions. Refer to the sections on these statements for further details.

The error message section is in numeric order. A short description follows each error number; a longer explanation is below. Following the longer explanation is a possible cause of the error and a recovery method.

**Error #** **Description**

**1** LINE TOO LONG

The program line exceeds 255 characters or the tokenized version of the program line exceeds 255 bytes.

Break the program line into 2 separate lines.

**2** OUT OF TABLE MEMORY

The number of table entries has exceeded the system limit. COMPSEE BASIC is limited in the number of variables, labels, and loops which may be in an application program. The limits are:

| Operating System | OSA | OSB |
|---|---|---|
| Variable names: 500 | 800 | |
| Labels: | 200 | 400 |
| Nested loops: | 11 | 11 |
| Nested subroutines: | 13 | 13 |
| IF-THEN-ELSE nest: | 6 | 6 |

"Loops" are FOR-NEXT loops and WHILE-WEND loops. The ONBAR and ONCOM statements are counted as nested subroutines when they are invoked. An IF-THEN-ELSE nest is when GOSUB is executed from within an IF statement. The following program has an IF-THEN-ELSE nest of 2:

```
10 IF A<>1 THEN GOSUB TEST: PRINT A ELSE PRINT B
20 STOP
100 TEST: IF A=2 THEN GOSUB TEST1 : PRINT C ELSE PRINT D RETURN
200 TEST1 : PRINT E : RETURN
```

**3** INVALID USE OF CHARACTERS

The program line has an invalid sequence of characters. Examples are:

An illegal double operator: = =, > >, or < <

A variable immediately followed by a quotation mark: ABLE"cain"

A decimal point within a line number: GOTO 30.2

Correct the program line.

**4** NAME TOO LONG

A variable or a label cannot be longer than eight characters.

Shorten the name to eight characters or less.

**5** INVALID CHARACTER

The character is not in the COMPSEE BASIC character set. Valid characters are between 32 and 126 decimal in the ASCII character set.

Eliminate the character.

| Error # | Description |
|---|---|
| **6** | UNMATCHED PARENTHESES<br>Every left parenthesis must have a matching right parenthesis; conversely, every right parenthesis must have a matching left parenthesis.<br>Match the parentheses |
| **7** | INVALID NAME<br>A variable name or a label may consist of alphanumerics only (i.e., A-Z and 0-9). The first character in the name must be a letter. The string designator '$ can only be at the end of a string variable name.<br>Correct the name. |
| **8** | INVALID NUMBER<br>The number is out of range of valid COMPSEE BASIC numbers. See the number section of this manual for the valid range of numbers. Another cause of this error is a non-numeric character encountered when a numeric is expected; for example, A + .B as opposed to A + .1.<br>Correct the number. |
| **9** | MISSING QUOTE<br>A string constant has no ending quotation (") mark.<br>Add the quotation mark. |
| **10** | INVALID LINE NUMBER<br>The program line has a line number greater than 65535 or equal to 0.<br>Correct the line number. |
| **11** | OUT OF PROGRAM MEMORY<br>The application program has exceeded the available memory in the Apex II device.<br>Memory may be freed by eliminating or shortening one or more RAM files or by shortening the application program. If the application program is extensively edited, issue a NEW command to the Apex II terminal and reenter the application program. An extensively edited program is fragmented in memory leaving sections of unused memory. The application program is stored contiguously when first entered. |
| **12** | UNDEFINED LINE NUMBER<br>The statement makes reference to a nonexistent line number.<br>Correct the statement. |
| **13** | INVALID SYNTAX<br>The program line contains an improper sequence of keywords, variables, or characters.<br>See Keyboard Commands (Section 5 of this part of the manual) for the correct syntax. |
| **14** | NUMBER OUT OF RANGE<br>The number is not in the valid range for the statement.<br>Check the proper section in this manual for the correct number range for the statement.<br>Another cause of this error is an invalid input parameter to a function. See the table at the end of this section for a list of COMPSEE BASIC functions and their possible error numbers. |
| **15** | CANNOT CONTINUE<br>An application program cannot be continued after the following:<br>• One or more program lines have been changed or deleted.<br>• An error 2 (Out of table memory) occurred.<br>• An error 11 (Out of program memory) occurred.<br>The program must be rerun. |

| Error # | Description |
|---|---|
| **16** | DUPLICATE DEFINITION<br>A previously dimensioned variable cannot be redefined; the variable is in a DIM statement twice. Unless a variable is explicitly dimensioned, it takes the COMPSEE BASIC defaults upon its first use. Numeric and string variables default to scalar. The default string length is 18.<br>Move all DIM statements to the start of the application program and check that no variables are dimensioned more than once. |
| **17** | ARRAY NOT DIMENSIONED<br>An array must be dimensioned in a DIM statement before being used.<br>Dimension the array. |
| **18** | INVALID USE OF A VARIABLE<br>A scalar variable is used as an array or an array variable is used as a scalar.<br>Correct the program line.<br>Another cause of this error is an attempt to assign a value to an element within an array that has not yet been dimensioned.<br>Properly dimension the array in question. |
| **19** | RECEIVE TIMEOUT<br>A delay of over DEVICE TIMEOUT seconds is encountered in receiving data via the RS-232-C port. See the SYSPARMS instruction in Section 16 of this part of the manual for an explanation of DEVICE TIMEOUT. Another cause of this error is a <ALT><C> detected from the keypad while receiving data.<br>Check the connection on the RS-232-C port and retry the operation. |
| **20** | PARITY ERROR<br>A parity error was detected or an attempt was made to send eight bits of data in seven bit mode.<br>Review the MODE statement to ensure the RS-232-C port is set up properly. Check the communication hardware and retry the statement. |
| **21** | INVALID RECORD LENGTH ON A LOAD COMMAND<br>The record length must be between 1 and 2047, inclusive.<br>Correct the length. |
| **22** | INVALID RECORD COUNT ON A LOAD COMMAND<br>The record count must be between 1 and 65535, inclusive.<br>Correct the count. |
| **23** | INVALID HEADER FORMAT ON A LOAD COMMAND<br>The header record contains a syntax error or a <ALT><C> was detected from the keypad during the header transmission.<br>Reenter the header information with the correct syntax. |
| **24** | TRANSMIT TIMEOUT<br>A delay of over DEVICE TIMEOUT seconds is encountered in transmitting data via the RS-232-C port. Refer to the SYSPARMS instruction in Section 16 of this part of the manual for an explanation of DEVICE TIMEOUT. Another cause of this error is an <ALT><C> detected from the keypad while transmitted. Use RESUME, not RESUME NEXT to recover from a Transmit Timeout unless another error also occurred.<br>Check the connection on the RS-232-C port and retry the operation. |

| Error # | Description |
|---|---|
| **25** | INVALID EXPRESSION<br>A non-arithmetic expression or non-character expression is encountered where an arithmetic expression or a character expression should be. For example:<br>`WRITE #10, USING "C2",REC=1,"HI"`<br>COMPSEE BASIC expects to see the character expression "HI" following the FORM field "C2". The REC = 1 causes an invalid expression error.<br>Correct the program line.<br>Another cause of this error is an invalid input parameter to a function. See the list of invalid function calls at the end of this section for a list of COMPSEE BASIC functions and their possible error numbers. |
| **26** | DIVIDE BY ZERO<br>The evaluation of an arithmetic expression results in a divide by zero.<br>Correct the program line. |
| **27** | STRING TOO LONG<br>The character string is too long to fit into the variable's defined maximum length, the character string is longer than 255 characters, or the character string does not fit into the specified FORM field.<br>Correct the program line. |
| **28** | OVERFLOW<br>This error is caused by one of the following conditions:<br>A. The number is greater than 9.9999999 E + 126.<br>B. The number is less than -9.9999999 E + 126.<br>C. The number does not fit into the specified FORM field.<br>Correct the program line. |
| **29** | UNDERFLOW<br>The number is less than 1 E-127 and greater than -1 E-127, but not equal to zero.<br>Correct the program line. |
| **30** | RESERVED |
| **31** | INVALID FUNCTION PARAMETERS<br>COMPSEE BASIC encountered a function with invalid parameters. For example, VAL(2), STR$("JOE"), LEN(9).<br>Correct the program line. |
| **32** | CANNOT DIMENSION A STRING TO ZERO<br>The valid range for a string dimension is 1 to 255, inclusive.<br>Correct the dimension. |
| **33** | NUMERIC STACK OVERFLOW<br>An arithmetic expression is nested too deeply, causing the COMPSEE BASIC numeric stack to overflow.<br>Simplify the arithmetic expression.<br>Another cause of this error is an invalid input parameter to a function. See the list of invalid function calls at the end of this section for a list of COMPSEE BASIC functions and their possible error numbers. |
| **34** | CHARACTER STACK OR OPERATOR STACK OVERFLOW<br>COMPSEE BASIC cannot evaluate a character or arithmetic expression because it is nested too deeply. |

| Error # | Description |
|---|---|
| **35** | ILLEGAL EXPRESSION<br>A character expression is encountered where a numeric expression is expected, or a numeric expression is encountered where a character expression is expected. Consider: `ABLE - "CAIN"`<br>This example tries to assign a character expression to a numeric variable.<br>Correct the program line.<br>Another cause of this error is an invalid input parameter to a function. See the table at the end of this section for a list of COMPSEE BASIC functions and their possible error numbers. |
| **36** | ARRAY OUT OF BOUNDS<br>The following are causes of this error:<br>1) An array with indices greater than the dimensioned maximum value, greater than 65534, or less than 0<br>2) A reference to an array element greater than 65534. |
| **37** | INVALID ARRAY REFERENCE<br>A one dimensional array is referenced as a two dimensional array, a two dimensional array is referenced as a one dimensional array, or an array is referenced with more than two dimensions.<br>Correct the program line. |
| **38** | UNDEFINED LABEL<br>A program line refers to a non-existent label. Refer to the label section in this manual for instructions on defining a label.<br>Define or eliminate the label. |
| **39** | DUPLICATE LABEL<br>The same label name is defined in two or more program lines. Refer to the label section in this manual.<br>Eliminate the duplication or all references to the label.<br>If the error has no associated line number, the label conflict is with an externally loaded label. |
| **40** | ELSE WITH NO THEN<br>Every ELSE clause of an IF-THEN-ELSE statement must have a matching THEN clause.<br>Correct the program line. |
| **41** | IF WITH NO THEN<br>Every IF statement must have a THEN clause.<br>Correct the program line. |
| **42** | FOR WITH NO TEXT<br>Every FOR loop must be terminated by a NEXT statement.<br>Correct the program line. |
| **43** | WHILE WITH NO WEND<br>Every WHILE loop must be terminated by a WEND statement.<br>Correct the program line. |
| **44** | FILE DOES NOT EXIST<br>A file name is referenced that does not exist in the directory.<br>Correct the program line. |

| Error # | Description |
|---|---|
| **45** | FILE NUMBER INCORRECT<br>A file number outside the allowable range is used. RAM files must have a value between 5 and 255, inclusive. The three system devices must use their reserved numbers:<br>0 = Screen<br>1 = Keypad buffer<br>4 = RS-232-C buffer<br>Correct the program line.<br>Another cause of this error is an invalid input parameter to a function. See the table at the end of this section for a list of COMPSEE BASIC functions and their possible error numbers. |
| **46** | FILE ALREADY OPEN<br>The file number used in the OPEN statement is already associated with another open file or the file name used in the OPEN statement is already opened under another file number.<br>Open the file once with an unused file number. |
| **47** | FILE RECORD LENGTH INVALID<br>The record length specified in an OPEN statement is invalid; the valid range is between 1 and 2047, inclusive.<br>Correct the length |
| **48** | FILE WITH SAME NAME ALREADY EXISTS<br>An attempt is made to OPEN a new RAM file with the same name as an existing file.<br>Open the new file under an unused name. |
| **49** | FILE NAME INVALID<br>A file name must be from one and eight characters long and contain any combination of characters from ASCII codes 35 and 126. A file name of "***" is also invalid (see the OPEN statement). |
| **50** | FILE NOT OPENED<br>An I/O statement cannot be used until the associated file is opened.<br>Open the file. |
| **51** | OUT OF FILE MEMORY<br>No more memory is available in the Apex II terminal for file storage.<br>Refer to error 11 for ways to free up memory. |
| **52** | TOO MANY FILES<br>The file directory is full. Only 40 files can exit in the Apex II terminal at one time.<br>Kill one or more existing files to free space in the file directory. |
| **53** | FORM STATEMENT OUT OF RANGE<br>The value after the FORM field is out of range or does not exist. Refer to the FORM section of this manual for the proper range values. An invalid FORM field without a value results in this error. An invalid FORM field followed by a value results in error 54.<br>Correct the FORM statement. |
| **54** | FORM STATEMENT SYNTAX ERROR<br>An invalid FORM field found in the FORM statement. Valid fields are P,N,E,C,W, and X. A FORM field used improperly also results in this error. For example, a W field cannot be used with a PRINT statement.<br>Correct the FORM statement. |
| **55** | MORE VARIABLES THAN FORM ELEMENTS<br>Every variable in an I/O statement list must have a corresponding FORM field.<br>Correct the FORM statement. |

| Error # | Description |
|---|---|
| **56** | READ PAST THE END OF A RECORD<br>The total of all the field lengths in a FORM statement cannot exceed the record length. For example, if file #10 has a record length of 20; the statement `READ #10, USING "P3,C5,E10.2,N5.1",A$,B,C` would result in this error. The total of all the field lengths is 20, however, the READ starts in position 3. Therefore, the N5.1 field attempts to go past the end of the record.<br>Correct the FORM statement. |
| **57** | WRITE PAST THE END OF A RECORD<br>This error is the same as error 56 except the I/O statement is a WRITE instead of a READ.<br>Correct the FORM statement. |
| **58** | RESUME WITHOUT ERROR<br>A RESUME statement recovers from an error condition. This error occurs if COMPSEE BASIC processes a RESUME and no error is pending. RESUME is used as a return from a user defined error handler. Check the program logic to see how the RESUME can be processed without an error pending. |
| **59** | RETURN WITHOUT GOSUB<br>A RETURN statement is the termination of a subroutine. This error occurs if a RETURN is processed without first processing a matching GOSUB.<br>Check the program logic to determine how a RETURN can be processed without a matching GOSUB. |
| **60** | CANNOT LOAD THE RAM FILE<br>An existing file must be closed before it can be overwritten via the LOAD command.<br>Close the file. |
| **61** | RESERVED |
| **62** | ILLEGAL DATE OR TIME<br>The date or time string is an illegal format. The time string must be in the form HH:MM:SS (leading zeros are not necessary); the date string must be in the form YY/MM/DD (leading zeros are not necessary).<br>Correct the program line. |
| **63** | ILLEGAL CURSOR POSITION<br>An INPUT statement cannot input a character if the cursor is off the screen. For example, the cursor is in position 4,19.<br>Correct the program line. |
| **64** | INVALID INPUT LENGTH<br>The length parameter on an INPUT statement must be between 1 and 255, inclusive.<br>Correct the program line. |
| **65** | TOO MUCH MEMORY ALLOCATED BY CLEAR OR INVALID BYTE COUNT<br>The CLEAR command can allocate memory for any user-defined Assembly routines. The amount allocated must be between 0 and 14688, inclusive. A second cause of this error is if the CLEAR is issued when an application program is already loaded in memory.<br>Correct the program line. |
| **66** | INVALID RESTORE LINE<br>A RESTORE statement points to a line that is not a DATA statement.<br>Correct the program line. |
| **67** | RESTORE WITH NO DATA STATEMENT<br>A RESTORE statement is processed in an application program that has no DATA statements.<br>Check the program logic to determine if a DATA statement is needed or if the RESTORE should be eliminated. |

| Error # | Description |
|---|---|

**68** NEXT WITH NO FOR

A NEXT statement is processed in an application program without an associated FOR statement being processed.

Check the program logic to determine if a FOR statement is needed or if the NEXT statement should be eliminated. Another possible cause of the error is one or more NEXT statements that are out of order. The following program causes this error:

```
10     FOR I = 1 TO 10
20     FOR J = 1 TO 10
30     NEXT I
40     NEXT J
```

**69** WEND WITH NO WHILE

A WEND statement is processed in an application program without an associated WHILE statement being processed.

Check the program logic to determine if a WHILE statement is needed or if the WEND statement should be eliminated.

**70** ILLEGAL MODE

The statement cannot be used in the present mode. For example, a STOP cannot be used in IMMEDIATE mode. Correct the program line.

**71** FORM FIELD DOES NOT MATCH DATA

The field specified in the form string does not match the corresponding data in the file record. For example, if a record written with the following statement:

```
WRITE #10, USING "C10", "ABCDEFGHIJ"
```

the following READ statement produces this error:

```
READ #10, USING "N5.0", REC = 1,A
```

Correct the form field.

**72** OUT OF DATA

No data remains for the READ statement.

Check the program logic to determine if another DATA statement is required, a RESTORE statement is required, or the READ statement has too many variables.

**73** OUT OF BUFFER SPACE

Buffer space is allocated as needed by the REWRITE statement. The size of the buffer space is equal to the length of the record being rewritten. This error occurs if the amount of unused memory is less than the record length.

Refer to error 11 for ways to free up memory.

**74** BREAK POINT TABLE FULL

The DIAGNOSTIC mode allows eight break points to be set at one time.

Clear the break table before setting the new break point. Refer to the DIAGNOSTIC section of this manual for further details.

**75** READ PAST END OF FILE

The following are causes of this error:

- A record number greater than the record count
- An invalid record number after the REC=
- A read without a REC = when the record pointer is on the last record

Correct the program line.

| Error # | Description |
|---|---|
| **76** | WRITE PAST END OF FILE<br>The following are causes of this error:<br>• A record number greater than the record count<br>• An invalid record number after the REC = in a REWRITE statement<br>• A WRITE where the record count goes past 65536.<br>Correct the program line. |
| **77-79** | RESERVED |
| **80** | SYSTEM LOCKED<br>The Apex II was locked using the LOCK statement and:<br>1) An attempt was made to give the Apex II a new lock code.<br>2) A LIST command was attempted.<br>Unlock the Apex II first. |
| **81** | INVALID LOCK/UNLOCK CODE<br>LOCK:<br>1) The UNLOCK code must be between 1 and 40 characters in length.<br>2) The UNLOCK code does not match the LOCK code.<br>Correct the LOCK/UNLOCK code. |
| **82** | SYSTEM CANNOT BE UNLOCKED<br>Eight unsuccessful UNLOCK attempts were made. The Apex II is locked until it is re-booted.<br>Reboot the Apex II. |
| **83** | CANNOT EDIT PROGRAM IN FLASH<br>The TRAKMSTR.EXE program was saved to FLASH and is therefore running from FLASH. It cannot be edited in flash.<br>Edit the program on the PC and then download the updated version using the Linkmstr.exe program. Alternatively, in the immediate mode, type >new and reload the program. |
| **84** | PROGRAM TOO LARGE TO SAVE IN FLASH<br>FLASH size is 64 K. A save was attempted with a program larger than 64 K.<br>Attempt to minimize the program by removing comment lines and editing. Try the operation again. |
| **85** | PROGRAM ALREADY SAVED IN FLASH<br>The user attempted to save a program to FLASH that was already saved in FLASH.<br>No action is necessary. |
| **86** | NO PROGRAM TO SAVE<br>The user has rebooted the Apex II and therefore both RAM and FLASH have been erased.<br>Reload the program from the PC using Linkmstr.exe. |

The following are severe COMPSEE BASIC errors. Try to rerun the program. If the error persists, reboot the Apex II terminal, reload the operating system (if necessary), and reload the application program.

| | |
|---|---|
| **200** | **INTERNAL STACK ERROR** |
| **252** | **CURSOR IN ILLEGAL LOCATION** |
| **253** | **INTERNAL LOAD ERROR** |
| **254** | **SCAN LOOP ERROR** |
| **255** | **INTERNAL EXECUTION ERROR** |

### *INVALID FUNCTION CALLS*

Some COMPSEE BASIC functions return error numbers which are not completely descriptive of the type of error which occurred. The error numbers have been left as is in order to retain compatibility with existing application programs which made use of these error numbers. The following is a list of invalid function calls and the error numbers which they return:

| | | | |
|---|---|---|---|
| ABS("string") | 31 | LOC(-1) | 31 |
| ASC(any number) | 35 | LOC(0) | 45 |
| CHR$("string") | 35 | LOC(2) | 45 |
| CHR$(-1) | 14 | LOC(3) | 45 |
| CHR$(256 and up) | 14 | LOC(256 and up) | 45 |
| EOF("string") | 45 | LOC(65536) | 31 |
| EOF(-1) | 31 | LOF("string") | 45 |
| EOF(0) | 45 | LOF(-1) | 31 |
| EOF(2) | 45 | LOF(0) | 45 |
| EOF(3) | 45 | LOF(2) | 45 |
| EOF(unallocated field #) | 50 | LOF(3) | 45 |
| EOF(256) | 45 | LOF(5 to 65535) | 45 |
| EOF(65536) | 31 | LOF(65536) | 31 |
| FILE$("string") | 35 | NOT("string") | 33 |
| FILE$(-1 to 4) | 45 | PEEK("string") | 31 |
| FILE$(256 and up) | 45 | PEEK(-1) | 14 |
| INSTR("string","string","string") | 45 | PEEK(top of memory+1) | 14 |
| INSTR(1,1,1) | 45 | RNUM("string") | 35 |
| INSTR("string","string") | 33 | RNUM(-1 to 4) | 45 |
| INSTR(1,"string") | 45 | RNUM(256 and up) | 45 |
| INSTR(-1,"string","string") | 31 | SGN("string") | 33 |
| INSTR(0,"string","string") | 31 | STR$("string") | 35 |
| INSTR(256 and up,"string","string")31 | 31 | STRING$("string") | 25 |
| INT("string") | 33 | STRING$(-1,-1) | 14 |
| LEN(any number) | 35 | STRING$(256 and up, 256 and up)14 | 14 |
| LOC("string") | 45 | VAL(any number) | 25 |

# 18 KEYPAD LOOKUP TABLE

COMPSEE BASIC uses a keypad table to assign characters to the keys. The table consists of the ASCII values of the characters. The character assigned to a key can be changed by modifying the value of the keypad table.

Valid ASCII values are 0 through 127. COMPSEE BASIC uses the high bit (value 128) to indicate whether the keypad stays in ALT function mode after a ALT function key is pressed. If a key in normal mode is assigned a value above 128, the actual value of the key is 128. For example, if the A key is assigned a value of 195, the key is assigned the letter C (195 - 128 = 67)).

The following example changes the ALT function A key on an alphanumeric keypad from a "$" to a "+".

```
POKE  24538,43
```

If the operator wants to leave the keypad in ALT function after the key is pressed, the statement is:

```
POKE  24538,171
```

| Normal Mode | | | ALT Function Mode | | |
|---|---|---|---|---|---|
| **Address** | **Character** | **Table Value** | **Address** | **Character** | **Table Value** |
| 24480 | 0 | 048 | 24528 | ALT - O | 015 |
| 24481 | 1 | 049 | 24529 | ALT - S | 147 |
| 24482 | 2 | 050 | 24530 | 2 | 178 |
| 24483 | 3 | 051 | 24531 | 3 | 179 |
| 24484 | 4 | 052 | 24532 | ALT - X | 152 |
| 24485 | 5 | 053 | 24533 | ALT - Y | 153 |
| 24486 | 6 | 054 | 24534 | 6 | 182 |
| 24487 | 7 | 055 | 24535 | ALT - R | 018 |
| 24488 | 8 | 056 | 24536 | ALT - V | 022 |
| 24489 | 9 | 057 | 24537 | ALT - Z | 026 |
| 24490 | A | 065 | 24538 | $ | 036 |
| 24491 | B | 066 | 24539 | = | 061 |
| 24492 | C | 067 | 24540 | ALT - C | 003 |
| 24493 | D | 068 | 24541 | ALT - D | 004 |
| 24494 | E | 069 | 24542 | Null | 000 |
| 24495 | F | 070 | 24543 | " | 034 |
| 24496 | G | 071 | 24544 | , | 044 |
| 24497 | H | 072 | 24545 | Null | 000 |
| 24498 | I | 073 | 24546 | Contrast Up | 011 |
| 24499 | J | 074 | 24547 | Contrast Down | 007 |
| 24500 | K | 075 | 24548 | ( | 040 |
| 24501 | L | 076 | 24549 | ) | 041 |
| 24502 | M | 077 | 24550 | Null | 000 |
| 24503 | N | 078 | 24551 | Null | 000 |
| 24504 | O | 079 | 24552 | Null | 000 |
| 24505 | P | 080 | 24553 | : | 058 |
| 24506 | Q | 081 | 24554 | ALT - Q | 017 |
| 24507 | R | 082 | 24555 | ALT - R | 018 |
| 24508 | S | 083 | 24556 | ALT - S | 019 |
| 24509 | T | 084 | 24557 | Null | 000 |
| 24510 | U | 085 | 24558 | Null | 000 |
| 24511 | V | 086 | 24559 | ALT - V | 022 |
| 24512 | W | 087 | 24560 | Null | 000 |
| 24513 | X | 088 | 24561 | ALT - X | 152 |
| 24514 | Y | 089 | 24562 | ALT - Y | 153 |
| 24515 | Z | 090 | 24563 | ALT - Z | 026 |
| 24516 | a | 097 | 24564 | e | 101 |
| 24517 | b | 098 | 24565 | f | 102 |
| 24518 | c | 099 | 24566 | g | 103 |
| 24519 | d | 100 | 24567 | h | 104 |
| 24520 | . (period) | 046 | 24568 | . | 106 |
| 24521 | Space | 032 | 24569 | Space | 032 |
| 24522 | Alt | 128 | 24570 | Alt | 128 |
| 24523 | Bksp | 127 | 24571 | Backspace | 136 |
| 24524 | Cursor Up | 030 | 24572 | Cursor Right | 144 |
| 24525 | Cursor Down | 031 | 24573 | Cursor Left | 157 |
| 24526 | Enter | 013 | 24574 | Enter | 141 |
| 24527 | Hardware Reserved | 000 | 24575 | Hardware Reserved | 000 |

# 19 ASCII TABLE

| ASCII Value | Character | Control Character | ASCII Value | Character | ASCII Value | Character |
|---|---|---|---|---|---|---|
| 000 | (Null) | NUL | 043 | + | 085 | U |
| 001 | | SOH | 044 | , | 086 | V |
| 002 | | STX | 045 | - | 087 | W |
| 003 | (ALT - C) | ETX | 046 | . | 088 | X |
| 004 | (ALT - D) | EOT | 047 | / | 089 | Y |
| 005 | | ENX | 048 | 0 | 090 | Z |
| 006 | | ACK | 049 | 1 | 091 | [ |
| 007 | | BEL | 050 | 2 | 092 | \ |
| 008 | (Backspace) | BS | 051 | 3 | 093 | ] |
| 009 | | HT | 052 | 4 | 094 | ^ |
| 010 | (Linefeed) | LF | 053 | 5 | 095 | - |
| 011 | | VT | 054 | 6 | 096 | ` |
| 012 | (Form Feed) | FF | 055 | 7 | 097 | a |
| 013 | (Carriage return) | CR | 056 | 8 | 098 | b |
| 014 | | SO | 057 | 9 | 099 | c |
| 015 | | SI | 058 | : | 100 | d |
| 016 | | DLE | 059 | ; | 101 | e |
| 017 | | DC1 | 060 | < | 102 | f |
| 018 | | DC2 | 061 | = | 103 | g |
| 019 | | DC3 | 062 | > | 104 | h |
| 020 | | DC4 | 063 | ? | 105 | i |
| 021 | | NAK | 064 | @ | 106 | j |
| 022 | | SYN | 065 | A | 107 | k |
| 023 | | ETB | 066 | B | 108 | l |
| 024 | | CAN | 067 | C | 109 | m |
| 025 | | EM | 068 | D | 110 | n |
| 026 | | SUB | 069 | E | 111 | o |
| 027 | | ESC | 070 | F | 112 | p |
| 028 | (Cursor right) | FS | 071 | G | 113 | q |
| 029 | (Cursor left) | GS | 072 | H | 114 | r |
| 030 | (Cursor up) | RS | 073 | I | 115 | s |
| 031 | (Cursor down) | US | 074 | J | 116 | t |
| 032 | (Space) | | 075 | K | 117 | u |
| 033 | ! | | 076 | L | 118 | v |
| 034 | " | | 077 | M | 119 | w |
| 035 | # | | 078 | N | 120 | x |
| 036 | $ | | 079 | O | 121 | y |
| 037 | % | | 080 | P | 122 | z |
| 038 | & | | 081 | Q | 123 | ( |
| 039 | ' | | 082 | R | 124 | |
| 040 | ( | | 083 | S | 125 | } |
| 041 | ) | | 084 | T | 126 | ~ |
| 042 | * | | | | 127 | (Delete) |

# 20 MEMORY MAP

TOP OF MEMORY

FFFFFH (1 Mb Unit)

7FFFFH (512K Unit)

3FFFFH (256K Unit)

1FFFFH (128K Unit)

| |
|---|
| **File Area**<br>(Grows Downward) |
| **Heap**<br>(Grows Toward Top of Memory) |
| **Program Area**<br>(Grows Toward Top of Memory) |
| **Basic Fixed Buffers** |
| **Monitor/Decoder Fixed Buffers** |
| **Operating System** |

The top of memory address depends on the amount of external RAM in the Apex II terminal.

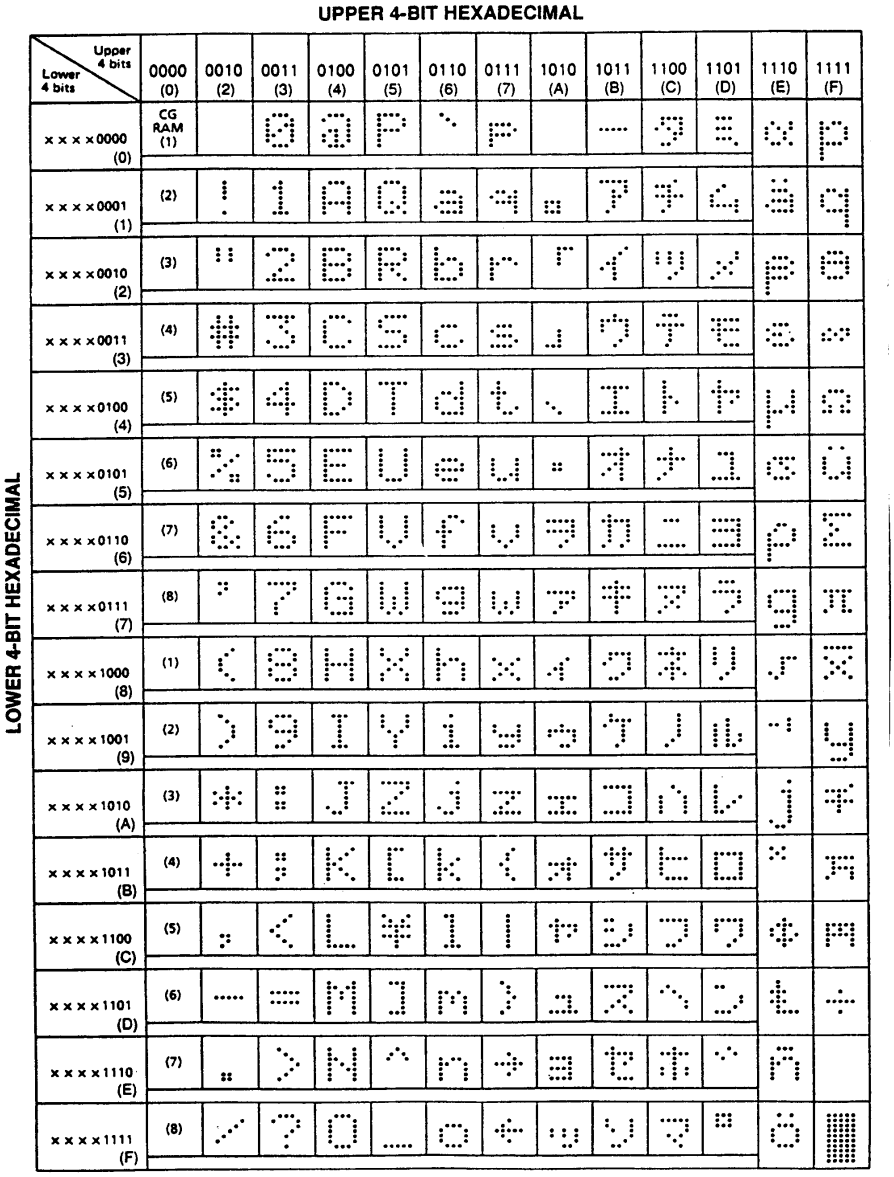
# 21 LCD SCREEN CHARACTER SET

The LCD screen can display 189 separate characters in any of its 64 positions; the character is generated using the CHR$ function. The code number of the character is calculated by adding the X and Y values taken from the table that follows.

For example, the following prints a summation sign to row 1, column 1 of the LCD screen:

LOCATE 1,1 : PRINT #0, USING “C1”, CHR$(246)

The code for a summation sign is 246 (X = 240, Y =6).



# 22 SAMPLE LOAD PROGRAMS

***BASIC Programs***

There are a number of sample BASIC programs available for the Apex II Terminals from the Compsee Inc. Bulletin Board Service (BBS) by dialing 910-439-1332. Sample programs are also provided in the Apex II Users Manual (Part 1 of this manual). Consult the Apex II Users Manual for directions on accessing the BBS, or to view sample programs.

Each of the tracking programs comes complete with instructions on what mode settings to use and how to load the program into the Apex II. Apex II demo software is license free and can be incorporated with your program in any form. If you have questions regarding these sample programs, please consult your Value Added Reseller (VAR) or call COMPSEE Customer Service at (407) 724-4321.

# 23 BREAK MODES

In normal operation, an COMPSEE BASIC device enters DIAGNOSTIC mode when one of the following program breaks is encountered within an application program:

1. A STOP statement
2. An error
3. A <ALT><C>, <ALT><D> sequence
4. A break point

Single stepping a program also causes COMPSEE BASIC to enter DIAGNOSTIC mode at the start of every line. COMPSEE BASIC checks the value at address 378 to determine how to handle a program break. This value determines two points:

1. where all error messages are sent and...
2. the mode COMPSEE BASIC enters after the break.

All other messages generated because of a program break are always sent to the screen of the COMPSEE device. For example, a STOP statement or a <ALT> <C>, <ALT> <D> sequence generates a BREAK line # message. This message is always sent to the screen. The default break value is zero. The following table describes the different modes:

| Value | Description |
|---|---|
| 0 | Error messages to the Apex II screen; enter DIAGNOSTIC mode. |
| 1 | Error messages to the Apex II screen; enter IMMEDIATE mode. |
| 2 | Error messages to the RS232-C port and the Apex II screen; enter IMMEDIATE mode. |
| 3 | Error messages to the RS232-C port; enter IMMEDIATE mode. |

If the break value is zero, a <ALT><C>, <ALT><D> sequence breaks the application program. If the break value is one, two, or three, a <ALT><C>, <ALT><D> sequence breaks the application program.

Normally, a <ALT><C>, <ALT><D> sequence (default break value of 0) breaks the application.

The break value can be changed by using the POKE statement: POKE378, value where value is number between zero and four, inclusive. A value greater than four gives unpredictable results. If the mode is changed (a zero value changed to a non-zero or a non-zero value changed to a zero), the Apex II terminal must be powered off and back on so COMPSEE BASIC can reset its internal flags.

# 24 COMMUNICATIONS

The Apex II terminals support RS232 asynchronous communications with other computers and peripherals. Communications can be either half duplex or full duplex; one start bit is always used. See the MODE instruction in Section 16 of this part of the manual for information on how to change communications parameters. The Apex II terminals have two communications buffers: transmit and receive. Each buffer has a fixed length of 254 bytes.

Transmission data is placed in the buffer and sent one byte at a time. If the buffer is full, BASIC attempts to add the byte to the buffer for a predetermined amount of time (set by the device timeout parameter in SYSPARMS). If the byte cannot be added to the buffer within that time, BASIC issues an error 24 (Transmit timeout). The PRINT and the PRINT #4, USING statements are used to put data into the transmit buffer. A <ALT> <C> detected from the Apex II terminal keypad during a PRINT statement also causes an error 24 and terminates the PRINT statement.

Data is received one byte at a time and stored in the buffer. The INPUT$ #4 statement is used to remove data from the buffer. If the buffer is empty, BASIC waits a predetermined amount of time (set by the device timeout parameter in SYSPARMS) for a byte. If a byte does not arrive in the buffer within that time, BASIC issues an error 19 (Receive timeout). A <ALT> <C> detected from the Apex II terminal keypad during the INPUT$ #4 statement also causes an error 19 and terminates the INPUT$ statement without updating the INPUT$ variable, however, no error is issued. <ALT> <C> termination can be disabled by the INP function.

### *BAUD RATES*

Seven baud rates are supported:

300
600
1200
2400
4800
9600
19200

### *DATA BITS AND PARITY*

The Apex II terminal communicates with any of the following:

- Seven data bits and odd parity
- Seven data bits and even parity
- Eight data bits and no parity

The initial power up setting is even parity with seven data bits.

Data can be transmitted and received in one of two modes; Binary or Hex ASCII. Binary form means that every byte is sent and received in ASCII format as eight bits, the number of data bits depending on whether parity is used. Hex ASCII mode means that every byte is sent and received in ASCII format as two Hex bytes, sixteen bits total. For example, the letter A is ASCII code 41H (01000001B). In Hex ASCII mode, the letter A is transmitted as an ASCII 4 (34H00110100B) and an ASCII 1 (31H00110001B). Conversely, in order to receive the letter A in Hex ASCII mode, the Apex II terminal must receive an ASCII 4 and an ASCII 1. All user communications should be in Binary mode; Hex ASCII mode is normally used for uploading new versions of BASIC.

A parity error (Error 20) occurs when the parity option is invoked and the Apex II device detects a parity error on a received byte ( in either data mode). An Error 20 also occurs when the parity option is invoked and the Apex II terminal attempts to transmit a byte with an ASCII code greater than 127D (i.e., the parity bit is set).

EXAMPLE:

```
10  MODE 2,3 `1200 baud, 7 data bits even parity XON/XOFF flow control
20  PRINT CHR$(131)
```

This example causes an error 20 when RUN.

## ***HANDSHAKING***

The Apex II terminal has a 10 pin modular connector. Six signals are brought out on the connector:

| | |
|---|---|
| Receive data | (Rx) |
| Transmit data | (Tx) |
| Data terminal ready | (DTR) |
| Clear to send | (CTS) |
| Request to send | (RTS) |
| Ground | |

When the Apex II terminal is turned on, DTR is set high and stays high as long as the unit is on. The RTS signal is normally high. The DSR signal is ignored by BASIC.

The flow of data between the Apex II terminal and an external device is controlled by one of two methods; RTS-CTS or XON-XOFF. Unlike most BASICs, flow control is built into COMPSEE BASIC Interpreter. The following tables show how the handshaking works:

Receive data

| | **RTS-CTS Mode** | **XON-XOFF Mode** |
|---|---|---|
| 1. Buffer contains less than 239 bytes | | RTS is high |
| 2. Buffer contains 239 bytes | RTS goes high | XOFF sent |
| 3. Buffer emptied from 239 bytes to 128 bytes | RTS goes high | XON sent |

The Apex II terminal receives data until the receive buffer is three bytes short of full. BASIC then tells the sending unit to suspend transmission. Once the buffer is half empty (i.e., the data is removed from the receive buffer via the INPUT$ #4 statement), BASIC tells the sending unit to re-enable transmission.

Transmit data

| | **RTS-CTS Mode** | **XON-XOFF Mode** |
|---|---|---|
| 1. Transmitting data and CTS goes low | Suspend data transmission | Continue data transmission |
| 2. Transmitting data and XOFF received | Ignore transmission | Suspend data transmission |
| 3. Transmitting data and CTS goes high | Continue data transmission | Continue data transmission |

When the Apex II terminal is in XON-XOFF mode, the control signals are ignored; therefore, XON-XOFF mode cannot detect if the Apex II terminal is connected to a peripheral. If the user is in XON-XOFF mode and transmits without the terminal connected to a peripheral, the data is lost.

The Apex II terminal can detect if it is connected to a peripheral while in RTS/CTS mode. The Apex II terminal does not have use of the DSR control line (the control line normally used to detect the presence of a peripheral); therefore, all the following rules apply:

1. The Apex II terminal is in RTS/CTS mode.
2. The transmit buffer is empty.
3. A character is about to be put into the transmit buffer (via the PRINT or PRINT #4 statement)
4. CTS is low (i.e., deasserted)

If the above conditions are true, BASIC gives an immediate error 24 (TRANSMIT TIMEOUT) as an indication the Apex II terminal is not connected to a peripheral.

As noted above, when CTS is low during normal data transmission (i.e., the transmit buffer already contains characters to transmit), BASIC waits a predetermined amount of time (specified by the device timeout parameter of the SYSPARMS statement) for CTS to go back high. If CTS does not go high within that period of time, BASIC aborts the transmission and no error is given.

A situation may arise where an error 24 is returned by BASIC during a normal data transfer. If the CTS line is lowered by the peripheral (because its receive buffer is full), and the Apex II terminal tries to put a character into its empty transmit buffer, BASIC will give an immediate error 24 because the four rules stated above are all true. BASIC also returns an error 24 if the transmit buffer of the Apex II terminal is full and BASIC has attempted to put a character into the buffer for device timeout seconds.

If an application program does RS-232-C communications, the communications error should always be trapped so the program can handle them appropriately. The communications errors are:

| | |
|---|---|
| RECEIVE TIMEOUT | 19 |
| PARITY ERROR | 20 |
| TRANSMIT TIMEOUT | 24 |

### ***Connecting The Apex II to Another Device***

The Apex II terminal can be connected to any computer that supports RS-232-C asynchronous communications. The cable that connects the two devices must have the following wire connections:

| **COMPSEE** | **Other Device** |
|---|---|
| TX pin | RX pin |
| RX pin | TX pin |
| CTS pin | RTS pin |
| RTS pin | CTS pin |
| Signal ground pin | Signal ground pin |

The communications ports of the Apex II and the other device must have the same configuration. On initial power up, the Apex II port is configured as: 9600 baud, 7 data bits, even parity, 1 start bit, 1 stop bit, and RTS/CTS flow control.

The Apex II terminal cannot be running a program if it is to communicate in IMMEDIATE mode with another device. See Section 23 - Break Modes, for information on how to break a program.

To ensure that the two devices are communicating, send a carriage return (ASCII code 13) and a linefeed (ASCII code 10) to the Apex II terminal. If the two devices are on-line, the terminal responds with the prompt character: >

### ***Communications While Scanning***

If the Apex II terminal must do communications while scanning (either with a wand or a laser scanner), RTS/CTS flow control must be used. BASIC always lowers the Apex II terminal's RTS line at the beginning of a scan and raises it at the end of a scan. BASIC also ensures that any character being received or transmitted at the start of the scan is fully completed.

***NOTE***

BASIC does not support XON/XOFF flow control while the Apex II terminal is scanning!!! If the Apex II terminal is in XON/XOFF flow control and the Apex II device receives characters after a scan has started, some characters will be lost. Regardless of the type of flow control in use, BASIC always suspends RS-232-C transmissions during a scan.

### ***Apex II Cabling Information***

| Signal Line | Apex II | XT DB 25 S | AT DB 9 | Modem DB 25 P | MAC* Circ 8 |
|---|---|---|---|---|---|
| Battery | 1 | NC | NC | NC | NC |
| Receive | 2 | 2 | 3 | 3 | 3 |
| Transmit | 3 | 3 | 2 | 2 | 5 |
| DTR | 4 | 6 | 6 | 20 | 1 |
| Ground | 5 | 7 | 5 | 7 | 4 |
| | 6 | NC | NC | NC | NC |
| RTS | 7 | 5 | 8 | 4 | NC |
| CTS | 8 | 4 | 7 | 5 | 2 |
| | 9 | NC | NC | NC | NC |
| | 10 | NC | NC | NC | NC |

***** It is advisable to use XON/XOFF Handshaking with the Macintosh computer. These pinouts are for the MAC+, MAC SE and MAC SE/30, CI, CX, etc.

Other Macintosh computer models are available from Apple. Please contact your Macintosh dealer for compatibility information.