Microsoft®
GW-BASIC® Interpreter

User's Guide

Microsoft Corporation

Document Number 410130001-320-R01-0686
Part Number 031-000-009

Contents

Figures

Tables

Chapter 1
Welcome to GW-BASIC

Microsoft® GW-BASIC® is a simple, easy-to-learn, easy-to-use computer programming language with English-like statements and mathematical notations. With GW-BASIC you will be able to write both simple and complex programs to run on your computer. You will also be able to modify existing software that is written in GW-BASIC.

This guide is designed to help you use the GW-BASIC Interpreter with the MS-DOS® operating system. Section 1.5 lists resources that will teach you how to program.

1.1 System Requirements

This version of GW-BASIC requires MS-DOS version 3.2 or later.

1.2 Preliminaries

Your GW-BASIC files will be on the MS-DOS diskette located at the back of the *MS-DOS User's Reference*. Be sure to make a working copy of the diskette before you proceed.

Note

This manual is written for the user familiar with the MS-DOS operating system. For more information on MS-DOS, refer to the *Microsoft MS-DOS 3.2 User's Guide* and *User's Reference.*

1.3 Notational Conventions

Throughout this manual, the following conventions are used to distinguish elements of text:

bold	Used for commands, options, switches, and literal portions of syntax that must appear exactly as shown.
italic	Used for filenames, variables, and placeholders that represent the type of text to be entered by the user.
`monospace`	Used for sample command lines, program code and examples, and sample sessions.
SMALL CAPS	Used for keys, key sequences, and acronyms.

Brackets surround optional command-line elements.

1.4 Organization of This Manual

The *GW-BASIC User's Guide* is divided into six chapters, nine appendixes, and a glossary:

Chapter 1, "Welcome to GW-BASIC," describes this manual.

Chapter 2, "Getting Started With GW-BASIC," is an elementary guideline on how to begin programming.

Chapter 3, "Reviewing and Practicing GW-BASIC," lets you use the principles of GW-BASIC explained in Chapter 2.

Chapter 4, "The GW-BASIC Screen Editor," discusses editing commands that can be used when inputting or modifying a GW-BASIC program. It also explains the unique properties of the ten redefinable function keys and of other keys and keystroke combinations.

Chapter 5, "Creating and Using Files," tells you how to create files and to use the diskette input/output (I/O) procedures.

Chapter 6, “Constants, Variables, Expressions, and Operators,” defines the elements of GW-BASIC and describes how you will use them.

Appendix A, “Error Codes and Messages,” is a summary of all the error codes and error messages that you might encounter while using GW-BASIC.

Appendix B, “Mathematical Functions,” describes how to calculate certain mathematical functions not intrinsic to GW-BASIC.

Appendix C, “ASCII Character Codes,” lists the ASCII character codes recognized by GW-BASIC.

Appendix D, “Assembly Language (Machine Code) Subroutines,” shows how to include assembly language subroutines with GW-BASIC.

Appendix E, “Converting BASIC Programs to GW-BASIC,” provides pointers on converting programs written in BASIC to GW-BASIC.

Appendix F, “Communications,” describes the GW-BASIC statements required to support RS-232 asynchronous communications with other computers and peripheral devices.

Appendix G, “Hexadecimal Equivalents,” lists decimal and binary equivalents to hexadecimal values.

Appendix H, “Key Scan Codes,” lists and illustrates the key scan code values used in GW-BASIC.

Appendix I, “Characters Recognized by GW-BASIC,” describes the GW-BASIC character set.

The Glossary defines words and phrases commonly used in GW-BASIC and data processing.

1.5 Bibliography

This manual is a guide to the use of the GW-BASIC Interpreter: it makes no attempt to teach the BASIC programming language. The following texts may be useful for those who wish to learn BASIC programming:

Albrecht, Robert L., LeRoy Finkel, and Jerry Brown. *BASIC*. 2d ed. New York: Wiley Interscience, 1978.

Coan, James. *Basic BASIC*. Rochelle Park, N.J.: Hayden Book Company, 1978.

Dwyer, Thomas A. and Margot Critchfield. *BASIC and the Personal Computer*. Reading, Mass.: Addison-Wesley Publishing Co., 1978.

Ettlin, Walter A. and Gregory Solberg. *The MBASIC Handbook*. Berkeley, Calif.: Osborne/McGraw Hill, 1983.

Knecht, Ken. *Microsoft BASIC*. Portland, Oreg.: Dilithium Press, 1982.

Chapter 2
Getting Started With GW-BASIC

This chapter describes how to load GW-BASIC into your system. It also explains the two different types of operation modes, line formats, and the various elements of GW-BASIC.

2.1 Loading GW-BASIC

To use the GW-BASIC language, you must load it into the memory of your computer from your working copy of the MS-DOS diskette. Use the following procedure:

1. Turn on your computer.
2. Insert your working copy of the MS-DOS diskette into Drive A of your computer, and press RETURN.
3. Type the following command after the A> prompt, and press RETURN:

   ```
   gwbasic
   ```

Once you enter GW-BASIC, the GW-BASIC prompt, *Ok*, will replace the MS-DOS prompt, *A>*.

On the screen, the line *XXXXX Bytes Free* indicates how many bytes are available for use in memory while using GW-BASIC.

The function key (F1–F10) assignments appear on the bottom line of the screen. These function keys can be used to eliminate key strokes and save you time. Chapter 4, "The GW-BASIC Screen Editor," contains detailed information on function keys.

2.2 Modes of Operation

Once GW-BASIC is initialized (loaded), it displays the *Ok* prompt. *Ok* means GW-BASIC is at *command level*; that is, it is ready to accept commands. At this point, GW-BASIC may be used in either of two modes: *direct mode* or *indirect mode.*

2.2.1 Direct Mode

In the direct mode, GW-BASIC statements and commands are executed as they are entered. Results of arithmetic and logical operations can be displayed immediately and/or stored for later use, but the instructions themselves are lost after execution. This mode is useful for debugging and for using GW-BASIC as a calculator for quick computations that do not require a complete program.

2.2.2 Indirect Mode

The indirect mode is used to enter programs. Program lines are always preceded by line numbers, and are stored in memory. The program stored in memory is executed by entering the RUN command.

2.3 The GW-BASIC Command Line Format

The GW-BASIC command line lets you change the environment or the conditions that apply while using GW-BASIC.

Note

When you specify modifications to the operating environment of GW-BASIC, be sure to maintain the parameter sequence shown in the syntax statement. To skip a parameter, insert a comma. This will let the computer know that you have no changes to that particular parameter.

GW-BASIC uses a command line of the following form:

gwbasic[*filename*][<*stdin*][[>]>*stdout*][/**f**:*n*][/**i**][/**s**:*n*][/**c**:*n*][/**m**:[*n*][,*n*]][/**d**]

filename is the name of a GW-BASIC program file. If this parameter is present, GW-BASIC proceeds as if a RUN command had been given. If no extension is provided for the filename, a default file extension of .BAS is assumed. The .BAS extension indicates that the file is a GW-BASIC file. The maximum number of characters a filename may contain is eight with a decimal and three extension characters.

<*stdin* redirects GW-BASIC's standard input to be read from the specified file. When used, it must appear before any switches.

This might be used when you have multiple files that might be used by your program and you wish to specify a particular input file.

>*stdout* redirects GW-BASIC's standard output to the specified file or device. When used, it must appear before any switches. Using >> before *stdout* causes output to be appended.

GW-BASIC can be redirected to read from standard input (keyboard) and write to standard output (screen) by providing the input and output filenames on the command line as follows:

gwbasic *program name <input file*[>]>*output file*

An explanation of file redirection follows this discussion of the GW-BASIC command line.

Switches appear frequently in command lines; they designate a specified course of action for the command, as opposed to using the default for that setting. A switch parameter is preceded by a slash (/).

/**f**:*n* sets the maximum number of files that may be opened simultaneously during the execution of a GW-BASIC program. Each file requires 194 bytes for the File Control Block (FCB) plus 128 bytes for the data buffer. The data buffer size may be altered with the /**s**: switch. If the /**f**: switch is omitted, the maximum number of open files defaults to 3. This switch is ignored unless the /**i** switch is also specified on the command line.

/**i** makes GW-BASIC statically allocate space required for file operations, based on the /**s** and /**f** switches.

/**s**:*n* sets the maximum record length allowed for use with files. The record length option in the OPEN statement cannot exceed this value. If the /**s**: switch is omitted, the record length defaults to 128 bytes. The maximum record size is 32767.

/**c**:*n* controls RS-232 communications. If RS-232 cards are present, /c:0 disables RS-232 support, and any subsequent I/O attempts for each RS-232 card present. If the /**c**: switch is omitted, 256 bytes are allocated for the receive buffer and 128 bytes for the transmit buffer for each card present.

The /**c**: switch has no affect when RS-232 cards are not present. The /**c**:*n* switch allocates *n* bytes for the receive buffer and 128 bytes for the transmit buffer for each RS-232 card present.

/**m**:*n*[,*n*] sets the highest memory location (first *n*) and maximum block size (second *n*) used by GW-BASIC. GW-BASIC attempts to allocate 64K bytes of memory for the data and stack segments. If machine language subroutines are to be used with GW-BASIC programs, use the /**m**: switch to set the highest location that GW-BASIC can use. The maximum block size is in multiples of 16. It is used to reserve space for user programs (assembly language subroutines) beyond GW-BASIC's workspace.

The default for maximum block size is the highest memory location. The default for the highest memory location is 64K bytes unless maximum block size is specified, in which case the default is the maximum block size (in multiples of 16).

/**d** allows certain functions to return double-precision results. When the /**d** switch is specified, approximately 3000 bytes of additional code space are used. The functions affected are ATN, COS, EXP, LOG, SIN, SQR, and TAN.

Note

All switch numbers may be specified as decimal, octal (preceded by &O), or hexadecimal (preceded by &H).

Sample GW-BASIC command lines are as follows:

The following uses 64K bytes of memory and three files; loads and executes the program file *payroll.bas*:

```
A>gwbasic PAYROLL
```

The following uses 64K bytes of memory and six files; loads and executes the program file *invent.bas*:

```
A>gwbasic INVENT /F:6
```

The following disables RS-232 support and uses only the first 32K bytes of memory. 32K bytes above that are reserved for user programs:

```
A>gwbasic /C:0 /M:32768,4096
```

The following uses four files and allows a maximum record length of 512 bytes:

```
A>gwbasic /F:4 /S:512
```

The following uses 64K bytes of memory and three files. Allocates 512 bytes to RS-232 receive buffers and 128 bytes to transmit buffers, and loads and executes the program file *tty.bas*:

```
A>gwbasic TTY /C:512
```

For more information about RS-232 Communications, see Appendix F.

Redirection of Standard Input and Output

When redirected, all INPUT, LINE INPUT, INPUT$, and INKEY$ statements are read from the specified input file instead of the keyboard.

All PRINT statements write to the specified output file instead of the screen.

Error messages go to standard output and to the screen.

File input from KYBD: is still read from the keyboard.

File output to SCRN: still outputs to the screen.

GW-BASIC continues to trap keys when the ON KEY *n* statement is used.

Typing CTRL-BREAK when output is redirected causes GW-BASIC to close any open files, issue the message “Break in line *nnnn*” to standard output, exit GW-BASIC, and return to MS-DOS.

When input is redirected, GW-BASIC continues to read from this source until a CTRL-Z is detected. This condition can be tested with the end-of-file (EOF) function. If the file is not terminated by a CTRL-Z, or if a GW-BASIC file input statement tries to read past the end of file, then any open files are closed, and GW-BASIC returns to MS-DOS.

For further information about these statements and other statements, functions, commands, and variables mentioned in this text, refer to the *GW-BASIC User's Reference.*

Some examples of redirection follow.

```
GWBASIC MYPROG >DATA.OUT
```

Data read by the INPUT and LINE INPUT statements continues to come from the keyboard. Data output by the PRINT statement goes into the *data.out* file.

```
gwbasic MYPROG <DATA.IN
```

Data read by the INPUT and LINE INPUT statements comes from *data.in.* Data output by PRINT continues to go to the screen.

```
gwbasic MYPROG <MYINPUT.DAT >MYOUTPUT.DAT
```

Data read by the INPUT and LINE INPUT statements now comes from the file *myinput.dat,* and data output by the PRINT statements goes into *myoutput.dat.*

```
gwbasic MYPROG <\SALES\JOHN\TRANS.DAT >>\SALES\SALES.DAT
```

Data read by the INPUT and LINE INPUT statements now comes from the file *\sales\john\trans.dat.* Data output by the PRINT statement is appended to the file *\sales\sales.dat.*

2.4 GW-BASIC Statements, Functions, Commands, and Variables

A GW-BASIC program is made up of several elements: keywords, commands, statements, functions, and variables.

2.4.1 Keywords

GW-BASIC keywords, such as **print**, **goto**, and **return** have special significance for the GW-BASIC Interpreter. GW-BASIC interprets keywords as part of statements or commands.

Keywords are also called *reserved words*. They cannot be used as variable names, or the system will interpret them as commands. However, keywords may be embedded within variable names.

Keywords are stored in the system as *tokens* (1- or 2-byte characters) for the most efficient use of memory space.

2.4.2 Commands

Commands and statements are both executable instructions. The difference between commands and statements is that commands are generally executed in the direct mode, or command level of the interpreter. They usually perform some type of program maintenance such as editing, loading, or saving programs. When GW-BASIC is invoked and the GW-BASIC prompt, *Ok*, appears, the system assumes command level.

2.4.3 Statements

A statement, such as ON ERROR...GOTO, is a group of GW-BASIC keywords generally used in GW-BASIC program lines as part of a program. When the program is run, statements are executed when, and as, they appear.

2.4.4 Functions

The GW-BASIC Interpreter performs both numeric and string functions.

2.4.4.1 Numeric Functions

The GW-BASIC Interpreter can perform certain mathematical (arithmetical or algebraic) calculations. For example, it calculates the sine (**sin**), cosine (**cos**), or tangent (**tan**) of angle *x*.

Unless otherwise indicated, only integer and single-precision results are returned by numeric functions.

2.4.4.2 String Functions

String functions operate on strings. For example, TIME$ and DATE$ return the time and date known by the system. If the current time and date are entered during system start-up, the correct time and date are given (the internal clock in the computer keeps track).

2.4.4.3 User-Defined Functions

Functions can be user-defined by means of the DEF FN statement. These functions can be either string or numeric.

2.4.5 Variables

Certain groups of alphanumeric characters are assigned values and are called *variables*. When variables are built into the GW-BASIC program they provide information as they are executed.

For example, ERR defines the latest error which occurred in the program; ERL gives the location of that error. Variables can also be defined and/or redefined by the user or by program content.

All GW-BASIC commands, statements, functions, and variables are individually described in the *GW-BASIC User's Reference*.

2.5 Line Format

Each of the elements of GW-BASIC can make up sections of a program that are called *statements*. These statements are very similar to sentences in English. Statements are then put together in a logical manner to create programs. The *GW-BASIC User's Reference* describes all of the statements available for use in GW-BASIC.

In a GW-BASIC program, lines have the following format:

nnnnn statement[*statements*]

nnnnn is a line number.

statement is a GW-BASIC statement.

A GW-BASIC program line always begins with a line number and must contain at least one character, but no more than 255 characters. Line numbers indicate the order in which the program lines are stored in memory, and are also used as references when branching and editing. The program line ends when you press the RETURN key.

Depending on the logic of your program, there may be more than one statement on a line. If so, each must be separated by a colon (:). Each of the lines in a program should be preceded by a line number. This number may be any whole integer from 0 to 65529. It is customary to use line numbers such as 10, 20, 30, and 40, in order to leave room for any additional lines that you may wish to include later. Since the computer will run the statements in numerical order, additional lines needn't appear in consecutive order on the screen: for example, if you entered line 35 after line 60, the computer would still run line 35 after line 30 and before line 40. This technique may save your reentering an entire program in order to include one line that you have forgotten.

The width of your screen is 80 characters. If your statement exceeds this width, the cursor will wrap to the next screen line automatically. Only when you press the RETURN key will the computer acknowledge the end of the line. Resist the temptation to press RETURN as you approach the edge of the screen (or beyond). The computer will automatically wrap the line for you. You can also press CTRL-RETURN, which causes the cursor to move to the beginning of the next screen line without actually entering the line. When you press RETURN, the entire logical line is passed to GW-BASIC for storage in the program.

In GW-BASIC, any line of text that begins with a numeric character is considered a program line and is processed in one of three ways after the RETURN key is pressed:

- A new line is added to the program. This occurs if the line number is legal (within the range of 0 through 65529), and if at least one alpha or special character follows the line number in the line.
- An existing line is modified. This occurs if the line number matches the line number of an existing line in the program. The existing line is replaced with the text of the newly-entered line. This process is called *editing*.

Note

> Reuse of an existing line number causes all of the information contained in the original line to be lost. Be careful when entering numbers in the indirect mode. You may erase some program lines by accident.

- An existing line is deleted. This occurs if the line number matches the line number of an existing line, and the entered line contains only a line number. If an attempt is made to delete a nonexistent line, an "Undefined line number" error message is displayed.

2.6 Returning to MS-DOS

Before you return to MS-DOS, you must save the work you have entered under GW-BASIC, or the work will be lost.

To return to MS-DOS, type the following after the *Ok* prompt, and press RETURN:

```
system
```

The system returns to MS-DOS, and the *A>* prompt appears on your screen.

Chapter 3
Reviewing and Practicing GW-BASIC

The practice sessions in this chapter will help you review what you have learned. If you have not done so, this is a good time to turn on your computer and load the GW-BASIC Interpreter.

3.1 Example for the Direct Mode

You can use your computer in the direct mode to perform fundamental arithmetic operations. GW-BASIC recognizes the following symbols as arithmetic operators:

Operation	**GW-BASIC Operator**
Addition	+
Subtraction	-
Multiplication	*
Division	/

To enter a problem, respond to the *Ok* prompt with a question mark (?), followed by the statement of the problem you want to solve, and press the RETURN key. In GW-BASIC, the question mark can be used interchangeably with the keyword PRINT. The answer is then displayed.

Type the following and press the RETURN key:

```
?2+2
```

GW-BASIC will display the answer on your screen:

```
?2+2
4
Ok
```

To practice other arithmetic operations, replace the + sign with the desired operator.

The GW-BASIC language is not restricted to arithmetic functions. You can also enter complex algebraic and trigonometric functions. The formats for these functions are provided in Chapter 6, "Constants, Variables, Expressions and Operators."

3.2 Examples for the Indirect Mode

The GW-BASIC language can be used for functions other than simple algebraic calculations. You can create a program that performs a series of operations and then displays the answer. To begin programming, you create lines of instructions called *statements*. Remember that there can be more than one statement on a line, and that each line is preceded by a number.

For example, to create the command PRINT 2+3 as a statement, type the following:

```
10 print 2+3
```

When you press the RETURN key, the cursor shifts to the next line, but nothing else happens. To make the computer perform the calculation, type the following and press the RETURN key:

```
run
```

Your screen should look like this:

```
Ok
10 print 2+3
run
 5
Ok
```

You have just written a program in GW-BASIC.

The computer reserves its calculation until specifically commanded to continue (with the RUN command). This allows you to enter more lines of instruction. When you type the RUN command, the computer does the addition and displays the answer.

The following program has two lines of instructions. Type it in:

```
10 x=3
20 print 2+x
```

Now use the RUN command to have the computer calculate the answer.

Your screen should look like this:

```
Ok
10 x=3
20 print 2+x
run
 5
Ok
```

The two features that distinguish a program from a calculation are

1. the numbered lines
2. the use of the RUN command

These features let the computer know that all the statements have been typed and the computation can be carried out from beginning to end. It is the numbering of the lines that first signals the computer that this is a program, not a calculation, and that it must not do the actual computation until the RUN command is entered.

In other words, calculations are done under the direct mode. Programs are written under the indirect mode.

To display the entire program again, type the LIST command and press the RETURN key:

```
list
```

Your screen should look like this:

```
Ok
10 x=3
20 print 2+x
run
Ok
 5
Ok
list
10 X=3
20 PRINT 2+X
Ok
```

You'll notice a slight change in the program. The lowercase letters you entered have been converted into uppercase letters. The LIST command makes this change automatically.

3.3 Function Keys

Function keys are keys that have been assigned to frequently-used commands. The ten function keys are located on the left side of your keyboard. A guide to these keys and their assigned commands appears on the bottom of the GW-BASIC screen. To save time and keystrokes, you can press a function key instead of typing a command name.

For example, to list your program again, you needn't type the LIST command; you can use the function key assigned to it, instead:

1. Press the F1 key.
2. Press RETURN.

Your program should appear on the screen.

To run the program, simply press the F2 key, which is assigned to the RUN command.

As you learn more commands, you'll learn how to use keys F3 through F10. Chapter 4, "The GW-BASIC Screen Editor," contains more information about keys used in GW-BASIC.

3.4 Editing Lines

There are two basic ways to change lines. You can

- delete and replace them
- alter them with the EDIT command

To delete a line, simply type the line number and press the RETURN key. For example, if you type *12* and press the RETURN key, line number 12 is deleted from your program.

To use the EDIT command, type the command EDIT, followed by the number of the line you want to change. For example, type the following and press the RETURN key:

```
edit 10
```

You can then use the following keys to perform editing:

Key	**Function**
CURSOR UP CURSOR DOWN CURSOR LEFT CURSOR RIGHT	Moves the cursor within the statement
BACKSPACE	Deletes the character to the left of the cursor
DELETE (DEL)	Deletes the current character
INSERT (INS)	Lets you insert characters to the left of the cursor

For example, to modify statement (line) 10 to read *x=4*, use the cursor-right control key to move the cursor under the 3, and then type a 4. The number 4 replaces the number 3 in the statement.

Now press the RETURN key, and then the F2 key.

Your screen displays the following:

```
Ok
10 X=4
RUN
 6
Ok
```

3.5 Saving Your Program File

Creating a program is like creating a data file. The program is a file that contains specific instructions, or statements, for the computer. In order to use the program again, you must save it, just as you would a data file.

To save a file in GW-BASIC, use the following procedure:

1. Press the F4 key.

   The command word *SAVE"* appears on your screen.

2. Type a name for the program, and press the RETURN key. The file is saved under the name you specified.

To recall a saved file, use the following procedure:

1. Press the F3 key.

   The command load *LOAD"* appears on your screen.
2. Type the name of the file.
3. Press RETURN.

The file is loaded into memory, and ready for you to list, edit, or run.

Chapter 4
The GW-BASIC Screen Editor

You can edit GW-BASIC program lines as you enter them, or after they have been saved in a program file.

4.1 Editing Lines in New Files

If an incorrect character is entered as a line is being typed, it can be deleted with the BACKSPACE or DEL keys, or with CTRL-H. After the character is deleted, you can continue to type on the line.

The ESC key lets you delete a line that is in the process of being typed. In other words, if you have not pressed the RETURN key, and you wish to delete the current line of entry, press the ESC key.

To delete the entire program currently residing in memory, enter the NEW command. NEW is usually used to clear memory prior to entering a new program.

4.2 Editing Lines in Saved Files

After you have entered your GW-BASIC program and saved it, you may discover that you need to make some changes. To make these modifications, use the LIST statement to display the program lines that are affected:

1. Reload the program.
2. Type the LIST command, or press the F1 key.
3. Type the line number, or range of numbers, to be edited.

The lines will appear on your screen.

4.2.1 Editing the Information in a Program Line

You can make changes to the information in a line by positioning the cursor where the change is to be made, and by doing one of the following:

- Typing over the characters that are already there.

- Deleting characters to the left of the cursor, using the BACKSPACE key.
- Deleting characters at the cursor position using the DEL key on the number pad.
- Inserting characters at the cursor position by pressing the INS key on the number pad. This moves the characters following the cursor to the right making room for the new information.
- Adding to or truncating characters at the end of the program line.

If you have changed more than one line, be sure to press RETURN on each modified line. The modified lines will be stored in the proper numerical sequence, even if the lines are not updated in numerical order.

Note

A program line will not actually have changes recorded within the GW-BASIC program until the RETURN key is pressed with the cursor positioned somewhere on the edited line.

You do not have to move the cursor to the end of the line before pressing the RETURN key. The GW-BASIC Interpreter remembers where each line ends, and transfers the whole line, even if RETURN is pressed while the cursor is located in the middle or at the beginning of the line.

To truncate, or cut off, a line at the current cursor position, type CTRL-END or CTRL-E, followed by pressing the RETURN key.

If you have originally saved your program to a program file, make sure that you save the edited version of your program. If you do not do this, your modifications will not be recorded.

4.3 Special Keys

The GW-BASIC Interpreter recognizes nine of the numeric keys on the right side of your keyboard. It also recognizes the BACKSPACE key, ESC key, and the CTRL key. The following keys and key sequences have special functions in GW-BASIC:

Key	**Function**
BACKSPACE or CTRL-H	Deletes the last character typed, or deletes the character to the left of the cursor. All characters to the right of the cursor are moved left one position. Subsequent characters and lines within the current logical line are moved up as with the DEL key.
CTRL-BREAK or CTRL-C	Returns to the direct mode, without saving changes made to the current line. It will also exit auto line-numbering mode.
CTRL-CURSOR-LEFT or CTRL-B	Moves the cursor to the beginning of the previous word. The previous word is defined as the next character to the left of the cursor in the set A to Z or in the set 0 to 9.
CTRL-CURSOR-RIGHT or CTRL-F	Moves the cursor to the beginning of the next word. The next word is defined as the next character to the right of the cursor in the set A to Z or in the set 0 to 9. In other words, the cursor moves to the next number or letter after a blank or other special character.
CURSOR-DOWN or CTRL--	Moves the cursor down one line on the screen.
CURSOR-LEFT or CTRL-]	Moves the cursor one position left. When the cursor is advanced beyond the left edge of the screen, it will wrap to the right side of the screen on the preceding line.
CURSOR-RIGHT or CTRL-\	Moves the cursor one position right. When the cursor is advanced beyond the right edge of the screen, it will wrap to the left side of the screen on the following line.
CURSOR-UP or CTRL-6	Moves the cursor up one line on the screen.
CTRL-BACKSPACE or DEL	Deletes the character positioned over the cursor. All characters to the right of the one deleted are then moved one position left to fill in where the deletion was made. If a logical line extends beyond one physical line, characters on subsequent lines are moved left one position to fill in the previous space, and the character in the first column of each subsequent line is moved up to the end of the preceding line.

	DEL (delete) is the opposite of INS (insert). Deleting text reduces logical line length.
CTRL-END or CTRL-E	Erases from the cursor position to the end of the logical line. All physical screen lines are erased until the terminating RETURN is found.
CTRL-N or END	Moves the cursor to the end of the logical line. Characters typed from this position are added to the line.
CTRL-RETURN or CTRL-J	Moves the cursor to the beginning of the next screen line. This lets you create logical program lines which are longer than the physical screen width. Logical lines may be up to 255 characters long. This function may also be used as a line feed.
CTRL-M or RETURN	Enters a line into the GW-BASIC program. It also moves the cursor to the next logical line.
CTRL-[or ESC	Erases the entire logical line on which the cursor is located.
CTRL-G	Causes a beep to emit from your computer's speaker.
CTRL-K or HOME	Moves the cursor to the upper left corner of the screen. The screen contents are unchanged.
CTRL-HOME or CTRL-L	Clears the screen and positions the cursor in the upper left corner of the screen.
CTRL-R or INS	Turns the Insert Mode on and off. Insert Mode is indicated by the cursor blotting the lower half of the character position. In Graphics Mode, the normal cursor covers the whole character position. When Insert Mode is active, only the lower half of the character position is blanked by the cursor. When Insert Mode is off, characters typed replace existing characters on the line. The SPACEBAR erases the character at the current cursor position and moves the cursor one character to the right. The CURSOR-RIGHT key moves the cursor one character to the right, but does not delete the character.

When Insert Mode is off, pressing the TAB key moves the cursor over characters until the next tab stop is reached. Tab stops occur every eight character positions.

When Insert Mode is on, characters following the cursor are moved to the right as typed characters are inserted before them at the current cursor position. After each keystroke, the cursor moves one position to the right. Line wrapping is observed. That is, as characters move off the right side of the screen, they are inserted from the left on subsequent lines. Insertions increase logical line length.

When Insert Mode is on, pressing the TAB key causes blanks to be inserted from current cursor position to the next tab stop. Line wrapping is observed as above.

CTRL-NUM LOCK or CTRL-S	Places the computer in a pause state. To resume operation, press any other key.
CTRL-PRTSC	Causes characters printed on the screen to echo to the lineprinter (**lpt1:**). In other words, you will be printing what you type on the screen. Pressing CTRL-PRTSC a second time turns off the echoing of characters to **lpt1:**.
SHIFT-PRTSC	Sends the current screen contents to the printer, effectively creating a snapshot of the screen.
CTRL-I or TAB	Moves the cursor to the next tab stop. Tab stops occur every eight columns.

4.4 Function Keys

Certain keys or combinations of keys let you perform frequently-used commands or functions with a minimum number of keystrokes. These keys are called *function keys*.

The special function keys that appear on the left side of your keyboard can be temporarily redefined to meet the programming requirements and specific functions that your program may require.

Function keys allow rapid entry of as many as 15 characters into a program with one keystroke. These keys are located on the left side of your keyboard and are labelled F1 through F10. GW-BASIC has already assigned special functions to each of these keys. You will notice that after you load GW-BASIC, these special key functions appear on the bottom line of your screen. These key assignments have been selected for you as some of the most frequently used commands.

Initially, the function keys are assigned the following special functions:

Table 4.1

GW-BASIC Function Key Assignments

Key	**Function**	**Key**	**Function**
F1	LIST	F6	,“LPT1:”<-
F2	RUN<-	F7	TRON<-
F3	LOAD”	F8	TROFF<-
F4	SAVE”	F9	KEY
F5	CONT<-	F10	SCREEN 0,0,0<-

Note

The <- following a function indicates that you needn’t press the RETURN key after the function key. The selected command will be immediately executed.

If you choose, you may change the assignments of these keys. Any one or all of the 10 function keys may be redefined. For more information, see the KEY and ON KEY statements in the *GW-BASIC User’s Reference.*

Chapter 5
Creating and Using Files

There are two types of files in MS-DOS systems:

- *Program files*, which contain the program or instructions for the computer
- *Data files*, which contain information used or created by program files

5.1 Program File Commands

The following are the commands and statements most frequently used with program files. The *GW-BASIC User's Reference* contains more information on each of them.

SAVE *filename*[**,a**][**,p**]

Writes to diskette the program currently residing in memory.

LOAD *filename*[**,r**]

Loads the program from a diskette into memory. LOAD deletes the current contents of memory and closes all files before loading the program.

RUN *filename*[**,r**]

Loads the program from a diskette into memory and runs it immediately. RUN deletes the current contents of memory and closes all files before loading the program.

MERGE *filename*

Loads the program from a diskette into memory, but does not delete the current program already in memory.

KILL *filename*

Deletes the file from a diskette. This command can also be used with data files.

NAME *old filename* **AS** *new filename*

Changes the name of a diskette file. Only the *name* of the file is changed. The file is not modified, and it remains in the same space and position on the disk. This command can also be used with data files.

5.2 Data Files

GW-BASIC programs can work with two types of data files:

- Sequential files
- Random access files

Sequential files are easier to create than random access files, but are limited in flexibility and speed when accessing data. Data written to a sequential file is a series of ASCII characters. Data is stored, one item after another (sequentially), in the order sent. Data is read back in the same way.

Creating and accessing random access files requires more program steps than sequential files, but random files require less room on the disk, because GW-BASIC stores them in a compressed format in the form of a string.

The following sections discuss how to create and use these two types of data files.

5.2.1 Creating a Sequential File

The following statements and functions are used with sequential files:

CLOSE	LOF
EOF	OPEN
INPUT#	PRINT#
LINE INPUT#	PRINT# USING
LOC	UNLOCK
LOCK	WRITE#

The following program steps are required to create a sequential file and access the data in the file:

1. Open the file in output (O) mode. The current program will use this file first for output:

   ```
   OPEN "O",#1,"filename"
   ```

2. Write data to the file using the PRINT# or WRITE# statement:

   ```
   PRINT#1,A$
   PRINT#1,B$
   PRINT#1,C$
   ```

3. To access the data in the file, you must close the file and reopen it in input (I) mode:

   CLOSE #1
   OPEN "I",#1,"*filename*"

4. Use the INPUT# or LINE INPUT# statement to read data from the sequential file into the program:

   INPUT#1,X$,Y$,Z$

Example 1 is a short program that creates a sequential file, *data*, from information input at the terminal.

Example 1

```
10 OPEN "O",#1,"DATA"
20 INPUT "NAME";N$
30 IF N$="DONE" THEN END
40 INPUT "DEPARTMENT";D$
50 INPUT "DATE HIRED";H$
60 PRINT#1,N$;",";D$",";H$
70 PRINT:GOTO 20
RUN
NAME? MICKEY MOUSE
DEPARTMENT? AUDIO/VISUAL AIDS
DATE HIRED? 01/12/72

NAME? SHERLOCK HOLMES
DEPARTMENT? RESEARCH
DATE HIRED? 12/03/65

NAME? EBENEEZER SCROOGE
DEPARTMENT? ACCOUNTING
DATE HIRED? 04/27/78

NAME? SUPER MANN
DEPARTMENT? MAINTENANCE
DATE HIRED? 08/16/78

NAME? DONE
OK
```

5.2.2 Accessing a Sequential File

The program in Example 2 accesses the file *data*, created in the program in Example 1, and displays the name of everyone hired in 1978.

Example 2

```
10 OPEN "I",#1,"DATA"
20 INPUT#1,N$,D$,H$
30 IF RIGHT$(H$,2)="78" THEN PRINT N$
40 GOTO 20
50 CLOSE #1
RUN
EBENEEZER SCROOGE
SUPER MANN
Input past end in 20
Ok
```

The program in Example 2 reads, sequentially, every item in the file. When all the data has been read, line 20 causes an "Input past end" error. To avoid this error, insert line 15, which uses the EOF function to test for end of file:

```
15 IF EOF(1) THEN END
```

and change line 40 to GOTO 15.

A program that creates a sequential file can also write formatted data to the diskette with the PRINT# USING statement. For example, the following statement could be used to write numeric data to diskette without explicit delimiters:

```
PRINT#1,USING"####.##,";A,B,C,D
```

The comma at the end of the format string serves to separate the items in the disk file.

The LOC function, when used with a sequential file, returns the number of 128-byte records that have been written to or read from the file since it was opened.

5.2.3 Adding Data to a Sequential File

When a sequential file is opened in O mode, the current contents are destroyed. To add data to an existing file without destroying its contents, open the file in append (A) mode.

The program in Example 3 can be used to create, or to add onto a file called *names*. This program illustrates the use of LINE INPUT. LINE INPUT will read in characters until it sees a carriage return indicator, or until it has read 255 characters. It does not stop at quotation marks or commas.

Example 3

```
10 ON ERROR GOTO 2000
20 OPEN "A",#1,"NAMES"
110 REM ADD NEW ENTRIES TO FILE
120 INPUT "NAME";N$
130 IF N$="" THEN 200 'CARRIAGE RETURN EXITS INPUT LOOP
140 LINE INPUT "ADDRESS? ";A$
150 LINE INPUT "BIRTHDAY? ";B$
160 PRINT#1,N$
170 PRINT#1,A$
180 PRINT#1,B$
190 PRINT:GOTO 120
200 CLOSE #1
2000 ON ERROR GOTO 0
```

In lines 10 and 2000 the ON ERROR GOTO statement is being used. This statement enables error trapping and specifies the first line (2000) of the error handling subroutine. Line 10 enables the error handling routine. Line 2000 disables the error handling routine and is the point where GW-BASIC branches to print the error messages.

5.3 Random Access Files

Information in random access files is stored and accessed in distinct, numbered units called *records*. Since the information is called by number, the data can be called from any disk location; the program needn't read the entire disk, as when seeking sequential files, to locate data. GW-BASIC supports large random files. The maximum logical record number is $2^{32} - 1$.

The following statements and functions are used with random files:

CLOSE	FIELD	MKI$
CVD	LOC	MKS$
CVI	LOCK	OPEN
CVS	LOF	PUT
EOF	LSET/RSET	UNLOCK
ET	MKD$	

5.3.1 Creating a Random Access File

The following program steps are required to create a random data file:

1. Open the file for random access (R) mode. The following example specifies a record length of 32 bytes. If the record length is omitted, the default is 128 bytes.

   OPEN "R",#1,"*filename*",32

2. Use the FIELD statement to allocate space in the random buffer for the variables that will be written to the random file:

   ```
   FIELD#1,20 AS N$,4 AS A$,8 AS P$
   ```

   In this example, the first 20 positions (bytes) in the random file buffer are allocated to the string variable N$. The next 4 positions are allocated to A$; the next 8 to P$.

3. Use LSET or RSET to move the data into the random buffer fields in left- or right-justified format (L = left SET; R = right SET). Numeric values must be made into strings when placed in the buffer. MKI$ converts an integer value into a string, MKS$ converts a single-precision value, and MKD$ converts a double-precision value.

   ```
   LSET N$=X$
   LSET A$=MKS$(AMT)
   LSET P$=TEL$
   ```

4. Write the data from the buffer to the diskette using the PUT statement:

   ```
   PUT #1,CODE%
   ```

The program in Example 4 takes information keyed as input at the terminal and writes it to a random access data file. Each time the PUT statement is executed, a record is written to the file. In the example, the 2-digit CODE% input in line 30 becomes the record number.

Note

Do not use a fielded string variable in an INPUT or LET statement. This causes the pointer for that variable to point into string space instead of the random file buffer.

Example 4

```
10 OPEN "R",#1,"INFOFILE",32
20 FIELD#1,20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "2-DIGIT CODE";CODE%
40 INPUT "NAME";X$
50 INPUT "AMOUNT";AMT
60 INPUT "PHONE";TEL$:PRINT
70 LSET N$=X$
80 LSET A$=MKS$(AMT)
90 LSET P$=TEL$
100 PUT #1,CODE%
110 GOTO 30
```

5.3.2 Accessing a Random Access File

The following program steps are required to access a random file:

1. Open the file in R mode:

   OPEN "R",#1,"*filename*",32

2. Use the FIELD statement to allocate space in the random buffer for the variables that will be read from the file:

   FIELD, #1, 20 AS N$, 4 AS A$, 8 AS P$

   In this example, the first 20 positions (bytes) in the random file buffer are allocated to the string variable N$. The next 4 positions are allocated to A$; the next 8 to P$.

   Note

   In a program that performs both INPUT and OUTPUT on the same random file, you can often use just one OPEN statement and one FIELD statement.

   ---

3. Use the GET statement to move the desired record into the random buffer:

   GET #1,CODE%

   The data in the buffer can now be accessed by the program.

4. Convert numeric values back to numbers using the convert functions: CVI for integers, CVS for single-precision values, and CVD for double-precision values.

   ```
   PRINT N$
   PRINT CVS(A$)
   .
   .
   .
   ```

The program in Example 5 accesses the random file, *infofile*, that was created in Example 4. By inputting the 3-digit code, the information associated with that code is read from the file and displayed.

Example 5

```
10 OPEN "R",#1,"INFOFILE",32
20 FIELD #1, 20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "2-DIGIT CODE";CODE%
40 GET #1, CODE%
50 PRINT N$
60 PRINT USING "$$###.##";CVS(A$)
70 PRINT P$:PRINT
80 GOTO 30
```

With random files, the LOC function returns the current record number. The current record number is the last record number used in a GET or PUT statement. For example, the following line ends program execution if the current record number in file#1 is higher than 99:

```
IF LOC(1)#99 THEN END
```

Example 6 is an inventory program that illustrates random file access. In this program, the record number is used as the part number, and it is assumed that the inventory will contain no more than 100 different part numbers.

Lines 900-960 initialize the data file by writing CHR$(255) as the first character of each record. This is used later (line 270 and line 500) to determine whether an entry already exists for that part number.

Lines 130-220 display the different inventory functions that the program performs. When you type in the desired function number, line 230 branches to the appropriate subroutine.

Example 6

```
120 OPEN"R",#1,"INVEN.DAT",39
125 FIELD#1,1 AS F$,30 AS D$, 2 AS Q$,2 AS R$,4 AS P$
130 PRINT:PRINT "FUNCTIONS:":PRINT
135 PRINT 1,"INITIALIZE FILE"
140 PRINT 2,"CREATE A NEW ENTRY"
150 PRINT 3,"DISPLAY INVENTORY FOR ONE PART"
160 PRINT 4,"ADD TO STOCK"
170 PRINT 5,"SUBTRACT FROM STOCK"
180 PRINT 6,"DISPLAY ALL ITEMS BELOW REORDER LEVEL"
220 PRINT:PRINT:INPUT"FUNCTION";FUNCTION
225 IF (FUNCTION<1)OR(FUNCTION#6) THEN PRINT "BAD FUNCTION
NUMBER":GOTO 130
230 ON FUNCTION GOSUB 900,250,390,480,560,680
240 GOTO 220
250 REM BUILD NEW ENTRY
260 GOSUB 840
270 IF ASC(F$) < # 255 THEN INPUT"OVERWRITE";A$:
 IF A$ < # "Y" THEN RETURN
280 LSET F$=CHR$(0)
290 INPUT "DESCRIPTION";DESC$
300 LSET D$=DESC$
310 INPUT "QUANTITY IN STOCK";Q%
320 LSET Q$=MKI$(Q%)
330 INPUT "REORDER LEVEL";R%
340 LSET R$=MKI$(R%)
350 INPUT "UNIT PRICE";P
360 LSET P$=MKS$(P)
370 PUT#1,PART%
380 RETURN
390 REM DISPLAY ENTRY
400 GOSUB 840
410 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
420 PRINT USING "PART NUMBER ###";PART%
430 PRINT D$
440 PRINT USING "QUANTITY ON HAND #####";CVI(Q$)
450 PRINT USING "REORDER LEVEL #####";CVI(R$)
460 PRINT USING "UNIT PRICE $$##.##";CVS(P$)
470 RETURN
480 REM ADD TO STOCK
490 GOSUB 840
500 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
510 PRINT D$:INPUT "QUANTITY TO ADD";A%
```

```
520 Q%=CVI(Q$)+A%
530 LSET Q$=MKI$(Q%)
540 PUT#1,PART%
550 RETURN
560 REM REMOVE FROM STOCK
570 GOSUB 840
580 IF ASC(F$)=255 THEN PRINT "NULL ENTRY":RETURN
590 PRINT D$
600 INPUT "QUANTITY TO SUBTRACT";S%
610 Q%=CVI(Q$)
620 IF (Q%-S%)<0 THEN PRINT "ONLY";Q%;" IN STOCK" :GOTO 600
630 Q%=Q%-S%
640 IF Q%= < CVI(R$) THEN PRINT "QUANTITY NOW";Q%;
"REORDER LEVEL";CVI(R$)
650 LSET Q$=MKI$(Q%)
660 PUT#1,PART%
670 RETURN
680 REM DISPLAY ITEMS BELOW REORDER LEVEL4
690 FOR I=1 TO 100
710 GET#1,I
720 IF CVI(Q$)<CVI(R$) THEN PRINT D$;" QUANTITY";
CVI(Q$) TAB(50) "REORDER LEVEL";CVI(R$)
730 NEXT I
740 RETURN
840 INPUT "PART NUMBER";PART%
850 IF(PART% < 1)OR(PART% # 100) THEN PRINT "BAD PART NUMBER":
GOTO 840 ELSE GET#1,PART%:RETURN
890 END
900 REM INITIALIZE FILE
910 INPUT "ARE YOU SURE";B$:IF B$ < # "Y" THEN RETURN
920 LSET F$=CHR$(255)
930 FOR I=1 TO 100
940 PUT#1,I
950 NEXT I
960 RETURN
```

Chapter 6

Constants, Variables, Expressions and Operators

After you have learned the fundamentals of programming in GW-BASIC, you will find that you will want to write more complex programs. The information in this chapter will help you learn more about the use of constants, variables, expressions, and operators in GW-BASIC, and how they can be used to develop more sophisticated programs.

6.1 Constants

Constants are static values the GW-BASIC Interpreter uses during execution of your program. There are two types of constants: *string* and *numeric*.

A *string constant* is a sequence of 0 to 255 alphanumeric characters enclosed in double quotation marks. The following are sample string constants:

"HELLO"
"$25,000.00"
"Number of Employees"

Numeric constants can be positive or negative. When entering a numeric constant in GW-BASIC, you should not type the commas. For instance, if the number *10,000* were to be entered as a constant, it would be typed as *10000*. There are five types of numeric constants: *integer*, *fixed-point*, *floating-point*, *hexadecimal*, and *octal*.

Constant	Description
Integer	Whole numbers between −32768 and +32767. They do not contain decimal points.
Fixed-Point	Positive or negative real numbers that contain decimal points.
Floating-Point Constants	Positive or negative numbers represented in exponential form (similar to scientific notation). A floating-point constant consists of an optionally-signed integer or fixed-point number (the mantissa), followed by the letter E and an optionally-signed integer (the exponent).

	The allowable range for floating-point constants is $3.0X10^{-39}$ to $1.7X10^{38}$. For example: 235.988E-7 = .0000235988 2359E6 = 2359000000
Hexadecimal	Hexadecimal numbers with prefix &H. For example: &H76 &H32F
Octal	Octal numbers with the prefix &O or &. For example: &O347 &1234

6.1.1 Single- and Double-Precision Form for Numeric Constants

Numeric constants can be integers, single-precision or double-precision numbers. Integer constants are stored as whole numbers only. Single-precision numeric constants are stored with 7 digits (although only 6 may be accurate). Double-precision numeric constants are stored with 17 digits of precision, and printed with as many as 16 digits.

A single-precision constant is any numeric constant with either

- seven or fewer digits
- exponential form using E
- a trailing exclamation point (!)

A double-precision constant is any numeric constant with either

- eight or more digits
- exponential form using D
- a trailing number sign (#)

The following are examples of single- and double-precision numeric constants:

Single-Precision Constants	Double-Precision Constants
46.8	345692811
−1.09E-06	−1.09432D-06
3489.0	3490.0#
22.5!	7654321.1234

6.2 Variables

Variables are the names that you have chosen to represent values used in a GW-BASIC program. The value of a variable may be assigned specifically, or may be the result of calculations in your program. If a variable is assigned no value, GW-BASIC assumes the variable's value to be zero.

6.2.1 Variable Names and Declarations

GW-BASIC variable names may be any length; up to 40 characters are significant. The characters allowed in a variable name are letters, numbers, and the decimal point. The first character in the variable name must be a letter. Special type declaration characters are also allowed.

Reserved words (all the words used as GW-BASIC commands, statements, functions, and operators) can't be used as variable names. However, if the reserved word is embedded within the variable name, it will be allowed.

Variables may represent either numeric values or strings.

6.2.2 Type Declaration Characters

Type declaration characters indicate what a variable represents. The following type declaration characters are recognized:

Character	Type of Variable
$	String variable
%	Integer variable
!	Single-precision variable
#	Double-precision variable

The following are sample variable names for each type:

Variable Type	Sample Name
String variable	N$
Integer variable	LIMIT%
Single-precision variable	MINIMUM!
Double-precision variable	Pl#

The default type for a numeric variable name is single-precision. Double-precision, while very accurate, uses more memory space and more calculation time. Single-precision is sufficiently accurate for most applications. However, the seventh significant digit (if printed) will not always be accurate. You should be very careful when making conversions between integer, single-precision, and double-precision variables.

The following variable is a single-precision value by default:

ABC

Variables beginning with FN are assumed to be calls to a user-defined function.

The GW-BASIC statements DEFINT, DEFSTR, DEFSNG, and DEFDBL may be included in a program to declare the types of values for certain variable names.

6.2.3 Array Variables

An *array* is a group or table of values referenced by the same variable name. Each element in an array is referenced by an *array variable* that is a subscripted integer or an integer expression. The subscript is enclosed within parentheses. An array variable name has as many subscripts as there are dimensions in the array.

For example,

```
V(10)
```

references a value in a one-dimensional array, while

```
T(1,4)
```

references a value in a two-dimensional array.

The maximum number of dimensions for an array in GW-BASIC is 255. The maximum number of elements per dimension is 32767.

Note

> If you are using an array with a subscript value greater than 10, you should use the DIM statement. Refer to the *GW-BASIC User's Reference* for more information. If a subscript greater than the maximum specified is used, you will receive the error message "Subscript out of range."

Multidimensional arrays (more than one subscript separated by commas) are useful for storing tabular data. For example, A(1,4) could be used to represent a two-row, five-column array such as the following:

```
Column          0       1       2       3       4

Row     0       10      20      30      40      50
Row     1       60      70      80      90      100
```

In this example, element A(1,2) = 80 and A(0,3) = 40.

Rows and columns begin with 0, not 1, unless otherwise declared. For more information, see the OPTION BASE statement in the *GW-BASIC User's Reference.*

6.2.4 Memory Space Requirements for Variable Storage

The different types of variables require different amounts of storage. Depending on the storage and memory capacity of your computer and the size of the program that you are developing, these can be important considerations.

Variable	Required Bytes of Storage
Integer	2
Single-precision	4
Double-precision	8

Arrays	Required Bytes of Storage
Integer	2 per element
Single-precision	4 per element
Double-precision	8 per element

Strings:

Three bytes overhead, plus the present contents of the string as one byte for each character in the string. Quotation marks marking the beginning and end of each string are not counted.

6.3 Type Conversion

When necessary, GW-BASIC converts a numeric constant from one type of variable to another, according to the following rules:

- If a numeric constant of one type is set equal to a numeric variable of a different type, the number is stored as the type declared in the variable name. For example:

  ```
  10 A% = 23.42
  20 PRINT A%
  RUN
   23
  ```

  If a string variable is set equal to a numeric value or vice versa, a "Type Mismatch" error occurs.

- During an expression evaluation, all of the operands in an arithmetic or relational operation are converted to the same degree of precision; that is, that of the most precise operand. Also, the result of an arithmetic operation is returned to this degree of precision. For example:

```
10 D# = 6#/7
20 PRINT D#
RUN
 .8571428571428571
```

The arithmetic is performed in double-precision, and the result is returned in D# as a double-precision value.

```
10 D = 6#/7
20 PRINT D
RUN
```

The arithmetic is performed in double-precision, and the result is returned to D (single-precision variable) rounded and printed as a single-precision value.

- Logical operators convert their operands to integers and return an integer result. Operands must be within the range of −32768 to 32767 or an "Overflow" error occurs.
- When a floating-point value is converted to an integer, the fractional portion is rounded. For example:

```
10 C% = 55.88
20 PRINT C%
RUN
 56
```

- If a double-precision variable is assigned a single-precision value, only the first seven digits (rounded) of the converted number are valid. This is because only seven digits of accuracy were supplied with the single-precision value. The absolute value of the difference between the printed double-precision number, and the original single-precision value, is less than 6.3E-8 times the original single-precision value. For example:

```
10 A = 2.04
20 B# = A
30 PRINT A;B#
RUN
 2.04  2.039999961853027
```

6.4 Expressions and Operators

An expression may be simply a string or numeric constant, a variable, or it may combine constants and variables with operators to produce a single value.

Operators perform mathematical or logical operations on values. The operators provided by GW-BASIC are divided into four categories:

- Arithmetic
- Relational
- Logical
- Functional

6.4.1 Arithmetic Operators

The following are the arithmetic operators recognized by GW-BASIC. They appear in order of precedence.

Operator	**Operation**
^	Exponentiation
-	Negation
*	Multiplication
/	Floating-point Division
+	Addition
-	Subtraction

Operations within parentheses are performed first. Inside the parentheses, the usual order of precedence is maintained.

The following are sample algebraic expressions and their GW-BASIC counterparts:

Algebraic Expression	BASIC Expression
$\frac{X-Z}{Y}$	(X-Y)/Z
$\frac{XY}{Z}$	X*Y/Z
$\frac{X+Y}{Z}$	(X+Y)/Z
$(X^2)^Y$	(X^2)^Y
X^{Y^Z}	X^(Y^Z)
$X(-Y)$	X*(-Y)

Two consecutive operators must be separated by parentheses.

6.4.1.1 Integer Division and Modulus Arithmetic

Two additional arithmetic operators are available: integer division and modulus arithmetic.

Integer division is denoted by the backslash (\). The operands are rounded to integers (must be within the range of −32768 to 32767) before the division is performed, and the quotient is truncated to an integer.

The following are examples of integer division:

```
10\4 = 2

25.68\6.99 = 3
```

In the order of occurrence within GW-BASIC, the integer division will be performed just after floating-point division.

Modulus arithmetic is denoted by the operator MOD. It gives the integer value that is the remainder of an integer division.

The following are examples of modulus arithmetic:

```
10.4 MOD 4 = 2
(10/4=2 with a remainder 2)

25.68 MOD 6.99 = 5
(26/7=3 with a remainder 5)
```

In the order of occurrence within GW-BASIC, modulus arithmetic follows integer division. The INT and FIX functions, described in the *GW-BASIC User's Reference*, are also useful in modulus arithmetic.

6.4.1.2 Overflow and Division by Zero

If, during the evaluation of an expression, a division by zero is encountered, the "Division by zero" error message appears, machine infinity with the sign of the numerator is supplied as the result of the division, and execution continues.

If the evaluation of an exponentiation results in zero being raised to a negative power, the "Division by Zero" error message appears, positive machine infinity is supplied as the result of the exponentiation, and execution continues.

If overflow occurs, the "Overflow" error message appears, machine infinity with the algebraically correct sign is supplied as the result, and execution continues. The errors that occur in overflow and division by zero will not be trapped by the error trapping function.

6.4.2 Relational Operators

Relational operators let you compare two values. The result of the comparison is either true (−1) or false (0). This result can then be used to make a decision regarding program flow.

Table 6.1 displays the relational operators.

Table 6.1

Relational Operators

Operator	Relation Tested	Expression
=	Equality	X=Y
<>	Inequality	X<>Y
<	Less than	X<Y
>	Greater than	X>Y
<=	Less than or equal to	X<=Y
>=	Greater than or equal to	X>=Y

The equal sign is also used to assign a value to a variable. See the LET statement in the *GW-BASIC User's Reference.*

When arithmetic and relational operators are combined in one expression, the arithmetic is always performed first:

```
X+Y < (T-1)/Z
```

This expression is true if the value of X plus Y is less than the value of T − 1 divided by Z.

6.4.3 Logical Operators

Logical operators perform tests on multiple relations, bit manipulation, or boolean operations. The logical operator returns a bit-wise result which is either true (not zero) or false (zero). In an expression, logical operations are performed after arithmetic and relational operations. The outcome of a logical operation is determined as shown in the following table. The operators are listed in order of precedence.

Table 6.2

Results Returned by Logical Operations

Operation	Value	Value	Result
NOT	X		NOT X
	T		F
	F		T
AND	X	Y	X AND Y
	T	T	T
	T	F	F
	F	T	F
	F	F	F

Table 6.2 *(continued)*

Operation	Value	Value	Result
OR	X	Y	X OR Y
	T	T	T
	T	F	T
	F	T	T
	F	F	F
XOR	X	Y	X XOR Y
	T	T	F
	T	F	T
	F	T	T
	F	F	F
EQV	X	Y	X EQV Y
	T	T	T
	T	F	F
	F	T	F
	F	F	T
IMP	X	Y	X IMP Y
	T	T	T
	T	F	F
	F	T	T
	F	F	T

Just as the relational operators can be used to make decisions regarding program flow, logical operators can connect two or more relations and return a true or false value to be used in a decision. For example:

```
IF D<200 AND F<4 THEN 80
IF I#10 OR K<0 THEN 50
IF NOT P THEN 100
```

Logical operators convert their operands to 16-bit, signed, two's complement integers within the range of −32768 to +32767. If the operands are not within this range, an error results. If both operands are supplied as 0 or

−1, logical operators return 0 or −1. The given operation is performed on these integers in bits; that is, each bit of the result is determined by the corresponding bits in the two operands.

Thus, it is possible to use logical operators to test bytes for a particular bit pattern. For instance, the AND operator may be used to mask all but one of the bits of a status byte at a machine I/O port. The OR operator may be used to merge two bytes to create a particular binary value. The following examples demonstrate how the logical operators work:

Example	Explanation
63 AND 16=16	63=binary 111111 and 16=binary 10000, so 63 AND 16=16
15 AND 14=14	15=binary 1111 and 14=binary 1110, so 15 AND 14=14 (binary 1110)
-1 AND 8=8	-1=binary 1111111111111111 and 8=binary 1000, so -1 AND 8=8
4 OR 2=6	4=binary 100 and 2=binary 10, so 4 OR 2=6 (binary 110)
10 OR 10=10	10=binary 1010, so 1010 OR 1010=1010 (10)
-1 OR -2=-1	-1=binary 1111111111111111 and -2=binary 1111111111111110,so -1 OR -2=-1. The bit complement of 16 zeros is 16 ones, which is the two's complement representation of -1.
NOT X=-(X+1)	The two's complement of any integer is the bit complement plus one.

6.4.4 Functional Operators

A function is used in an expression to call a predetermined operation that is to be performed on an operand. GW-BASIC has intrinsic functions that reside in the system, such as SQR (square root) or SIN (sine).

GW-BASIC also allows user-defined functions written by the programmer. See the DEF FN statement in the *GW-BASIC User's Reference.*

6.4.5 String Operators

To compare strings, use the same relational operators used with numbers:

Operator	Meaning
=	Equal to
<>	Unequal
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

The GW-BASIC Interpreter compares strings by taking one character at a time from each string and comparing their ASCII codes. If the ASCII codes in each string are the same, the strings are equal. If the ASCII codes differ, the lower code number will precede the higher code. If the interpreter reaches the end of one string during string comparison, the shorter string is said to be smaller, providing that both strings are the same up to that point. Leading and trailing blanks are significant.

For example:

```
"AA" < "AB"
"FILENAME" = "FILENAME"
"X&" # "X#"
"CL " # "CL"
"kg" # "KG"
"SMYTH" < "SMYTHE"
B$ < "9/12/78" where B$ = "8/12/78"
```

String comparisons can also be used to test string values or to alphabetize strings. All string constants used in comparison expressions must be enclosed in quotation marks.

Strings can be concatenated by using the plus (+) sign. For example:

```
10 A$="FILE":B$="NAME"
20 PRINT A$+B$
30 PRINT "NEW " + A$+B$
RUN
FILENAME
NEW FILENAME
```

Appendix A
Error Codes and Messages

Code:	**Message:**
1	`NEXT without FOR` NEXT statement does not have a corresponding FOR statement. Check variable at FOR statement for a match with the NEXT statement variable.
2	`Syntax error` A line is encountered that contains an incorrect sequence of characters (such as unmatched parentheses, a misspelled command or statement, incorrect punctuation). This error causes GW-BASIC to display the incorrect line in edit mode.
3	`RETURN without GOSUB` A RETURN statement is encountered for which there is no previous GOSUB statement.
4	`Out of DATA` A READ statement is executed when there are no DATA statements with unread data remaining in the program.
5	`Illegal function call` An out-of-range parameter is passed to a math or string function. An illegal function call error may also occur as the result of • a negative or unreasonably large subscript • a negative or zero argument with LOG • a negative argument to SQR • a negative mantissa with a noninteger power

- a call to a USR function for which the starting address has not yet been given
- an improper argument to MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, or ON...GOTO

6 `Overflow`

The result of a calculation is too large to be represented in GW-BASIC's number format. If underflow occurs, the result is zero, and execution continues without an error.

7 `Out of memory`

A program is too large, has too many FOR loops, GOSUBs, variables, or expressions that are too complicated. Use the CLEAR statement to set aside more stack space or memory area.

8 `Undefined line number`

A line reference in a GOTO, GOSUB, IF-THEN...ELSE, or DELETE is a nonexistent line.

9 `Subscript out of range`

An array element is referenced either with a subscript that is outside the dimensions of the array, or with the wrong number of subscripts.

10 `Duplicate Definition`

Two DIM statements are given for the same array, or a DIM statement is given for an array after the default dimension of 10 has been established for that array.

11 `Division by zero`

A division by zero is encountered in an expression, or the operation of involution results in zero being raised to a negative power. Machine infinity with the sign of the numerator is supplied as the result of the division, or positive machine infinity is supplied as the result of the involution, and execution continues.

12	`Illegal direct` A statement that is illegal in direct mode is entered as a direct mode command.
13	`Type mismatch` A string variable name is assigned a numeric value or vice versa; a function that expects a numeric argument is given a string argument or vice versa.
14	`Out of string space` String variables have caused GW-BASIC to exceed the amount of free memory remaining. GW-BASIC allocates string space dynamically until it runs out of memory.
15	`String too long` An attempt is made to create a string more than 255 characters long.
16	`String formula too complex` A string expression is too long or too complex. Break the expression into smaller expressions.
17	`Can't continue` An attempt is made to continue a program that • has halted because of an error • has been modified during a break in execution • does not exist
18	`Undefined user function` A USR function is called before the function definition (DEF statement) is given.
19	`No RESUME` An error-trapping routine is entered but contains no RESUME statement.

20	`RESUME without error`
	A RESUME statement is encountered before an error-trapping routine is entered.
21	`Unprintable error`
	No error message is available for the existing error condition. This is usually caused by an error with an undefined error code.
22	`Missing operand`
	An expression contains an operator with no operand following it.
23	`Line buffer overflow`
	An attempt is made to input a line that has too many characters.
24	`Device Timeout`
	GW-BASIC did not receive information from an I/O device within a predetermined amount of time.
25	`Device Fault`
	Indicates a hardware error in the printer or interface card.
26	`FOR Without NEXT`
	A FOR was encountered without a matching NEXT.
27	`Out of Paper`
	The printer is out of paper; or, a printer fault.
28	`Unprintable error`
	No error message is available for the existing error condition. This is usually caused by an error with an undefined error code.
29	`WHILE without WEND`
	A WHILE statement does not have a matching WEND.

30	`WEND without WHILE` A WEND was encountered without a matching WHILE.
31-49	`Unprintable error` No error message is available for the existing error condition. This is usually caused by an error with an undefined error code.
50	`FIELD overflow` A FIELD statement is attempting to allocate more bytes than were specified for the record length of a random file.
51	`Internal error` An internal malfunction has occurred in GW-BASIC. Report to your dealer the conditions under which the message appeared.
52	`Bad file number` A statement or command references a file with a file number that is not open or is out of range of file numbers specified at initialization.
53	`File not found` A LOAD, KILL, NAME, FILES, or OPEN statement references a file that does not exist on the current diskette.
54	`Bad file mode` An attempt is made to use PUT, GET, or LOF with a sequential file, to LOAD a random file, or to execute an OPEN with a file mode other than I, O, A, or R.
55	`File already open` A sequential output mode OPEN is issued for a file that is already open, or a KILL is given for a file that is open.
56	`Unprintable error` An error message is not available for the error condition which exists. This is usually caused by an error with an undefined error code.

57	`Device I/O Error`
	Usually a disk I/O error, but generalized to include all I/O devices. It is a fatal error; that is, the operating system cannot recover from the error.
58	`File already exists`
	The filename specified in a NAME statement is identical to a filename already in use on the diskette.
59-60	`Unprintable error`
	No error message is available for the existing error condition. This is usually caused by an error with an undefined error code.
61	`Disk full`
	All disk storage space is in use.
62	`Input past end`
	An INPUT statement is executed after all the data in the file has been input, or for a null (empty) file. To avoid this error, use the EOF function to detect the end of file.
63	`Bad record number`
	In a PUT or GET statement, the record number is either greater than the maximum allowed (16,777,215) or equal to zero.
64	`Bad filename`
	An illegal form is used for the filename with LOAD, SAVE, KILL, or OPEN; for example, a filename with too many characters.
65	`Unprintable error`
	No error message is available for the existing error condition. This is usually caused by an error with an undefined error code.
66	`Direct statement in file`
	A direct statement is encountered while loading a ASCII-format file. The LOAD is terminated.

67 `Too many files`

An attempt is made to create a new file (using SAVE or OPEN) when all directory entries are full or the file specifications are invalid.

68 `Device Unavailable`

An attempt is made to open a file to a nonexistent device. It may be that hardware does not exist to support the device, such as **lpt2**: or **lpt3**:, or is disabled by the user. This occurs if an OPEN "COM1: statement is executed but the user disables RS-232 support with the /**c**: switch directive on the command line.

69 `Communication buffer overflow`

Occurs when a communications input statement is executed, but the input queue is already full. Use an ON ERROR GOTO statement to retry the input when this condition occurs. Subsequent inputs attempt to clear this fault unless characters continue to be received faster than the program can process them. In this case several options are available:

- Increase the size of the COM receive buffer with the /**c**: switch.
- Implement a hand-shaking protocol with the host/satellite (such as: XON/XOFF, as demonstrated in the TTY programming example) to turn transmit off long enough to catch up.
- Use a lower baud rate for transmit and receive.

70 `Permission Denied`

This is one of three hard disk errors returned from the diskette controller.

- An attempt has been made to write onto a diskette that is write protected.
- Another process has attempted to access a file already in use.
- The UNLOCK range specified does not match the preceding LOCK statement.

71 `Disk not Ready`

Occurs when the diskette drive door is open or a diskette is not in the drive. Use an ON ERROR GOTO statement to recover.

72 `Disk media error`

Occurs when the diskette controller detects a hardware or media fault. This usually indicates damaged media. Copy any existing files to a new diskette and reformat the damaged diskette. FORMAT maps the bad tracks in the file allocation table. The remainder of the diskette is now usable.

73 `Advanced Feature`

An attempt was made to use a reserved word that is not available in this version of GW-BASIC.

74 `Rename across disks`

Occurs when an attempt is made to rename a file to a new name declared to be on a disk other than the disk specified for the old name. The naming operation is not performed.

75 `Path/File Access Error`

During an OPEN, MKDIR, CHDIR, or RMDIR operation, MS-DOS is unable to make a correct path-to-filename connection. The operation is not completed.

76 `Path not found`

During an OPEN, MKDIR, CHDIR, or RMDIR operation, MS-DOS is unable to find the path specified. The operation is not completed.

Appendix B
Mathematical Functions

Mathematical functions not intrinsic to GW-BASIC can be calculated as follows:

Function	**GW-BASIC Equivalent**
Secant	SEC(X) = 1/COS(X)
Cosecant	CSC(X) = 1/SIN(X)
Cotangent	COT(X) = 1/TAN(X)
Inverse Sine	ARCSIN(X) = ATN(X/SQR(-X*X + 1))
Inverse Cosine	ARCCOS(X) = ATN (X/SQR(-X*X + 1)) + π/2
Inverse Secant	ARCSEC(X) = ATN(X/SQR(X*X-1)) + SGN(SGN(X)-1)* π/2
Inverse Cosecant	ARCCSC(X) = ATN(X/SQR(X*X-1)) + SGN(X)-1)* π/2
Inverse Cotangent	ARCCOT(X) = ATN(X) + π/2
Hyperbolic Sine	SINH(X) = (EXP(X)-EXP(-X))/2
Hyperbolic Cosine	COSH(X) = (EXP(X) + EXP(-X))/2
Hyperbolic Tangent	TANH(X) = (EXP(X)-EXP(-X))/(EXP(X) + EXP(-X))
Hyperbolic Secant	SECH(X) = 2/(EXP(X) + EXP(-X))
Hyperbolic Cosecant	CSCH(X) = 2/(EXP(X)-EXP(-X))
Hyperbolic Cotangent	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))*2 + 1

Inverse Hyperbolic Sine	ARCSINH(X) = LOG(X/SQR(X*X + 1))
Inverse Hyperbolic Cosine	ARCCOSH(X) = LOG(X + SQR(X*X-1))
Inverse Hyperbolic Tangent	ARCTANH(X) = LOG((1 + X)/(1-X))/2
Inverse Hyperbolic Cosecant	ARCCSCH(X) = LOG(SGN(X)*SQR(X*X + 1) + 1)/X
Inverse Hyperbolic Secant	ARCSECH(X) = LOG(SQR(-X*X + 1) + 1)/X
Inverse Hyperbolic Cotangent	ARCCOTH(X) = LOG((X + 1)/(X-1))/2

Appendix C
ASCII Character Codes

Dec	Oct	Hex	Chr	Dec	Oct	Hex	Chr
000	000	00H	NUL	032	040	20H	SP
001	001	01H	SOH	033	041	21H	!
002	002	02H	STX	034	042	22H	"
003	003	03H	ETX	035	043	23H	#
004	004	04H	EOT	036	044	24H	$
005	005	05H	ENQ	037	045	25H	%
006	006	06H	ACK	038	046	26H	&
007	007	07H	BEL	039	047	27H	'
008	010	08H	BS	040	050	28H	(
009	011	09H	HT	041	051	29H	)
010	012	0AH	LF	042	052	2AH	*
011	013	0BH	VT	043	053	2BH	+
012	014	0CH	FF	044	054	2CH	,
013	015	0DH	CR	045	055	2DH	-
014	016	0EH	SO	046	056	2EH	.
015	017	0FH	SI	047	057	2FH	/
016	020	10H	DLE	048	060	30H	0
017	021	11H	DC1	049	061	31H	1
018	022	12H	DC2	050	062	32H	2
019	023	13H	DC3	051	063	33H	3
020	024	14H	DC4	052	064	34H	4
021	025	15H	NAK	053	065	35H	5
022	026	16H	SYN	054	066	36H	6
023	027	17H	ETB	055	067	37H	7
024	030	18H	CAN	056	070	38H	8
025	031	19H	EM	057	071	39H	9
026	032	1AH	SUB	058	072	3AH	:
027	033	1BH	ESC	059	073	3BH	;
028	034	1CH	FS	060	074	3CH	<
029	035	1DH	GS	061	075	3DH	=
030	036	1EH	RS	062	076	3EH	>
031	037	1FH	US	063	077	3FH	?

Dec = Decimal, Oct = Octal, Hex = Hexadecimal(H), Chr = Character, LF = Line feed
FF = Form feed, CR = Carriage return, DEL = Rubout

Appendix C *(continued)*

Dec	Oct	Hex	Chr	Dec	Oct	Hex	Chr
064	100	40H	@	096	140	60H	`
065	101	41H	A	097	141	61H	a
066	102	42H	B	098	142	62H	b
067	103	43H	C	099	143	63H	c
068	104	44H	D	100	144	64H	d
069	105	45H	E	101	145	65H	e
070	106	46H	F	102	146	66H	f
071	107	47H	G	103	147	67H	g
072	110	48H	H	104	150	68H	h
073	111	49H	I	105	151	69H	i
074	112	4AH	J	106	152	6AH	j
075	113	4BH	K	107	153	6BH	k
076	114	4CH	L	108	154	6CH	l
077	115	4DH	M	109	155	6DH	m
078	116	4EH	N	110	156	6EH	n
079	117	4FH	O	111	157	6FH	o
080	120	50H	P	112	160	70H	p
081	121	51H	Q	113	161	71H	q
082	122	52H	R	114	162	72H	r
083	123	53H	S	115	163	73H	s
084	124	54H	T	116	164	74H	t
085	125	55H	U	117	165	75H	u
086	126	56H	V	118	166	76H	v
087	127	57H	W	119	167	77H	w
088	130	58H	X	120	170	78H	x
089	131	59H	Y	121	171	79H	y
090	132	5AH	Z	122	172	7AH	z
091	133	5BH	[	123	173	7BH	{
092	134	5CH	\	124	174	7CH	|
093	135	5DH	]	125	175	7DH	}
094	136	5EH	^	126	176	7EH	~
095	137	5FH	_	127	177	7FH	DEL

Dec = Decimal, Oct = Octal, Hex = Hexadecimal(H), Chr = Character, LF = Line feed
FF = Form feed, CR = Carriage return, DEL = Rubout

Appendix D
Assembly Language (Machine Code) Subroutines

This appendix is written primarily for users experienced in assembly language programming.

GW-BASIC lets you interface with assembly language subroutines by using the USR function and the CALL statement.

The USR function allows assembly language subroutines to be called in the same way GW-BASIC intrinsic functions are called. However, the CALL statement is recommended for interfacing machine language programs with GW-BASIC. The CALL statement is compatible with more languages than the USR function call, produces more readable source code, and can pass multiple arguments.

D.1 Memory Allocation

Memory space must be set aside for an assembly language (or machine code) subroutine before it can be loaded. There are three recommended ways to set aside space for assembly language routines:

- Specify an array and use VARPTR to locate the start of the array before every access.
- Use the **/m** switch in the command line. Get GW-BASIC's Data segment (DS), and add the size of DS to reference the reserved space above the data segment.
- Execute a .COM file that stays resident, and store a pointer to it in an unused interrupt vector location.

There are three recommended ways to load assembly language routines:

- BLOAD the file. Use DEBUG to load in an .EXE file that is in high memory, run GW-BASIC, and BSAVE the .EXE file.
- Execute a .COM file that contains the routines. Save the pointer to these routines in unused interrupt-vector locations, so that your application in GW-BASIC can get the pointer and use the routine(s).
- Place the routine into the specified area.

If, when an assembly language subroutine is called, more stack space is needed, GW-BASIC stack space can be saved, and a new stack set up for use by the assembly language subroutine. The GW-BASIC stack space must be restored, however, before returning from the subroutine.

D.2 CALL Statement

CALL *variablename*[(*arguments*)]

variablename contains the offset in the current segment of the subroutine being called.

arguments are the variables or constants, separated by commas, that are to be passed to the routine.

For each parameter in *arguments*, the 2-byte offset of the parameter's location within the data segment (DS) is pushed onto the stack.

The GW-BASIC return address code segment (CS), and offset (IP) are pushed onto the stack.

A long call to the segment address given in the last DEF SEG statement and the offset given in *variablename* transfers control to the user's routine.

The stack segment (SS), data segment (DS), extra segment (ES), and the stack pointer (SP) must be preserved.

Figure D.1 shows the state of the stack at the time of the CALL statement:

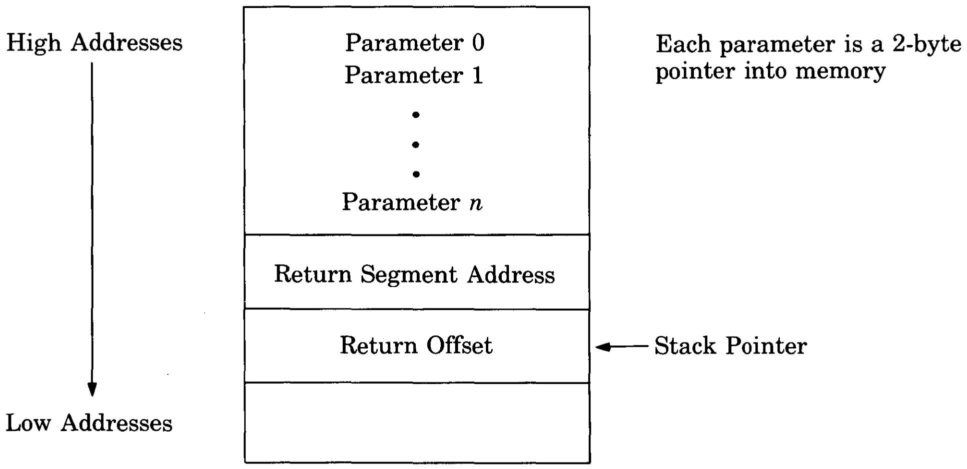

Figure D.1 Stack Layout When the CALL Statement is Activated

The user's routine now has control. Parameters may be referenced by moving the stack pointer (SP) to the base pointer (BP) and adding a positive offset to BP.

Upon entry, the segment registers DS, ES, and SS all point to the address of the segment that contains the GW-BASIC interpreter code. The code segment register CS contains the latest value supplied by DEF SEG. If no DEF SEG has been specified, it then points to the same address as DS, ES, and SS (the default DEF SEG).

Figure D.2 shows the condition of the stack during execution of the called subroutine:

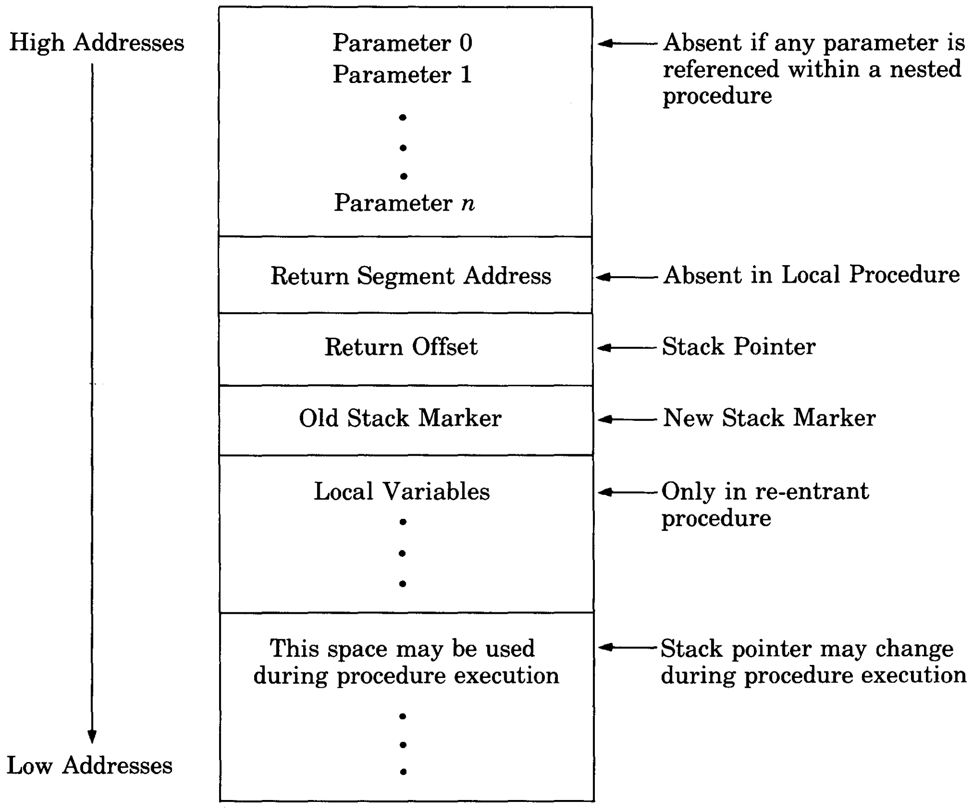
High Addresses
Low Addresses
Parameter 0
Parameter 1
Parameter *n*
Return Segment Address
Return Offset
Old Stack Marker
Local Variables
This space may be used during procedure execution
Absent if any parameter is referenced within a nested procedure
Absent in Local Procedure
Stack Pointer
New Stack Marker
Only in re-entrant procedure
Stack pointer may change during procedure execution

Figure D.2 Stack Layout During Execution of a CALL Statement

The following seven rules must be observed when coding a subroutine:

1. The called routine may destroy the contents of the AX, BX, CX, DX, SI, DI, and BP registers. They do not require restoration upon return to GW-BASIC. However, all segment registers and the stack pointer must be restored. Good programming practice dictates that interrupts enabled or disabled be restored to the state observed upon entry.

2. The called program must know the number and length of the parameters passed. References to parameters are positive offsets added to BP, assuming the called routine moved the current stack pointer into BP; that is, MOV BP,SP. When 3 parameters are passed, the location of PO is at BP+10, P1 is at BP+8, and P2 is at BP+6.
3. The called routine must do a RETURN *n* (*n* is two times the number of parameters in the argument list) to adjust the stack to the start of the calling sequence. Also, programs must be defined by a PROC FAR statement.
4. Values are returned to GW-BASIC by including in the argument list the variable name that receives the result.
5. If the argument is a string, the parameter offset points to three bytes called the *string descriptor.* Byte 0 of the string descriptor contains the length of the string (0 to 255). Bytes 1 and 2, respectively, are the lower and upper eight bits of the string starting address in string space.

Note

The called routine must not change the contents of any of the three bytes of the string descriptor.

6. Strings may be altered by user routines, but their length must not be changed. GW-BASIC cannot correctly manipulate strings if their lengths are modified by external routines.
7. If the argument is a string literal in the program, the string descriptor points to program text. Be careful not to alter or destroy your program this way. To avoid unpredictable results, add +"" to the string literal in the program. For example, the following line forces the string literal to be copied into string space allocated outside of program memory space:

```
20 A$="BASIC"+""
```

The string can then be modified without affecting the program.

Examples:

```
100 DEF SEG=&H2000
110 ACC=&H7FA
```

```
120 CALL ACC(A,B$,C)
.
.
.
```

Line 100 sets the segment to 2000 hex. The value of variable ACC is added into the address as the low word after the DEF SEG value is left-shifted four bits (this is a function of the microprocessor, not of GW-BASIC). Here, ACC is set to &H7FA, so that the call to ACC executes the subroutine at location 2000:7FA hex.

Upon entry, only 16 bytes (eight words) remain available within the allocated stack space. If the called program requires additional stack space, then the user program must reset the stack pointer to a new allocated space. Be sure to restore the stack pointer adjusted to the start of the calling sequence on return to GW-BASIC.

The following assembly language sequence demonstrates access of the parameters passed and storage of a return result in the variable C.

Note

The called program must know the variable type for numeric parameters passed. In these examples, the following instruction copies only two bytes:

```
MOVSW
```

This is adequate if variables A and C are integer. It would be necessary to copy four bytes if they were single precision, or copy eight bytes if they were double precision.

```
MOV BP,SP        Gets the current stack position in BP
MOV BX,8[BP]     Gets the address of B$ description
MOV CL,[BX]      Gets the length of B$ in CL
MOV DX,1[BX]     Gets the address of B$ string descriptor in D
MOV SI,10[BP]    Gets the address of A in SI
MOV DI,6[BP]     Gets the pointer to C in DI
MOVSW            Stores variable A in 'C'
RET 6            Restores stack; returns
```

D.3 USR Function Calls

Although the CALL statement is the recommended way of calling assembly language subroutines, the USR function call is still available for compatibility with previously-written programs.

Syntax:

USR[*n*](*argument*)

n is a number from 0 to 9 which specifies the USR routine being called (see DEF USR statement). If *n* is omitted, USR0 is assumed.

argument is any numeric or string expression.

In GW-BASIC a DEF SEG statement should be executed prior to a USR function call to ensure that the code segment points to the subroutine being called. The segment address given in the DEF SEG statement determines the starting segment of the subroutine.

For each USR function call, a corresponding DEF USR statement must have been executed to define the USR function call offset. This offset and the currently active DEF SEG address determine the starting address of the subroutine.

When the USR function call is made, register AL contains the *number type flag* (NTF), which specifies the type of argument given. The NTF value may be one of the following:

NTF Value	**Specifies**
2	a two-byte integer (two's complement format)
3	a string
4	a single-precision floating-point number
8	a double-precision floating-point number

If the argument of a USR function call is a number (AL<>73), the value of the argument is placed in the *floating-point accumulator* (FAC). The FAC is 8 bytes long and is in the GW-BASIC data segment. Register BX will point at the fifth byte of the FAC. Figure D.3 shows the representation of all the GW-BASIC number types in the FAC:

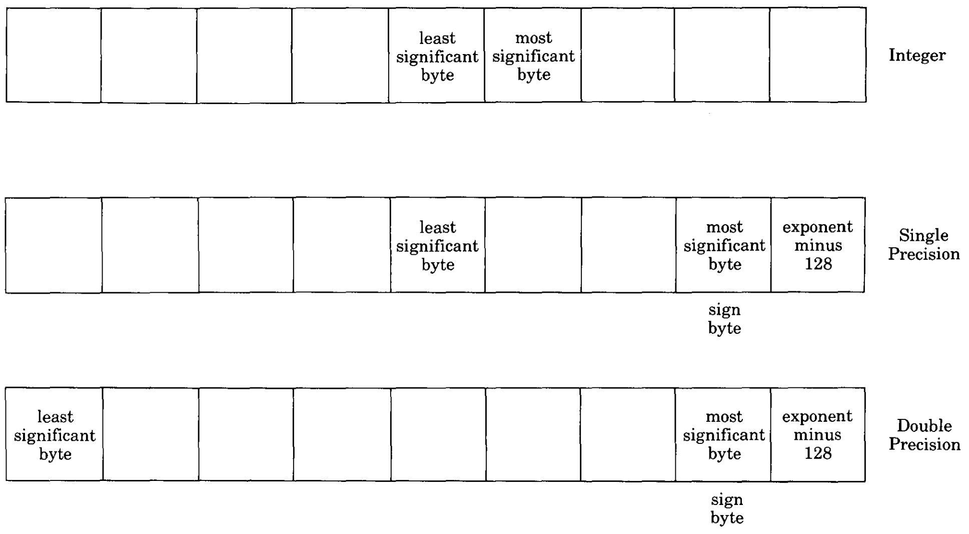

Figure D.3 Number Types in the Floating-Point Accumulator

If the argument is a single-precision floating-point number:

- BX+3 is the exponent, minus 128. The binary point is to the left of the most significant bit of the mantissa.
- BX+2 contains the highest seven bits of mantissa with leading 1 suppressed (implied). Bit 7 is the sign of the number (0=positive, 1=negative).
- BX+1 contains the middle 8 bits of the mantissa.
- BX+0 contains the lowest 8 bits of the mantissa.

If the argument is an integer:

- BX+1 contains the upper eight bits of the argument.
- BX+0 contains the lower eight bits of the argument.

If the argument is a double-precision floating-point number:

- BX+0 through BX+3 are the same as for single-precision floating point.
- BX-1 to BX-4 contain four more bytes of mantissa. BX-4 contains the lowest eight bits of the mantissa.

If the argument is a string (indicated by the value 3 stored in the AL register) the (DX) register pair points to three bytes called the string descriptor. Byte 0 of the string descriptor contains the length of the string (0 to 255). Bytes 1 and 2, respectively, are the lower- and upper-eight bits of the string starting address in the GW-BASIC data segment.

If the argument is a string literal in the program, the string descriptor points to program text. Be careful not to alter or destroy programs this way (see the preceding CALL statement).

Usually, the value returned by a USR function call is the same type (integer, string, single precision, or double precision) as the argument that was passed to it. The registers that must be preserved are the same as in the CALL statement.

A far return is required to exit the USR subroutine. The returned value must be stored in the FAC.

D.4 Programs That Call Assembly Language Programs

This section contains two sample GW-BASIC programs that

- load an assembly language routine to add two numbers together
- return the sum into memory
- remain resident in memory

The code segment and offset to the first routine is stored in interrupt vector at 0:100H.

Example 1 calls an assembly language subroutine:

Example 1

```
10 DEF SEG=0
100 CS=PEEK(&H102)+PEEK(&H103)*256
200 OFFSET=PEEK(&H100)+PEEK(&H101)*256
250 DEF SEG
```

```
300 C1%=2:C2%=3:C3%=0
400 TWOSUM=OFFSET
500 DEF SEG=CS
600 CALL TWOSUM(C1%,C2%,C3%)
700 PRINT C3%
800 END
```

The assembly language subroutine called in the above program must be assembled, linked, and converted to a .COM file. The program, when executed prior to the running of the GW-BASIC program, will remain in memory until the system power is turned off, or the system is rebooted.

```
0100                    org 100H
0100                    double    segment
                        assume  cs:double
0100    EB 17 90       start:          jmp      start1
0103                   usrprg   proc far
0103    55               push bp
0104    8B EC            mov bp,sp
0106    8B 76 08         mov si,[bp]+8          ;get address of
                                                ;parameter b
0109    8B 04            mov ax,[si]            ;get value of b
010B    8B 76 0A         mov si,[bp]+10         ;get address of
                                                ;parameter a
010E    03 04            add ax,[si]            ;add value of
                                                ;a to value of
                                                ;b
0110    8B 7E 06         mov di,[bp]+6          ;get address of
                                                ;parameter c
0113    89 05            mov di,ax              ;store sum in
                                                ;parameter c

0115    5D               pop bp
0116    ca 0006          ret 6
0119                   usrprg endp
                                                ;
                                                ;Program to put
                                                ;procedurein memory
                                                ;and remain resident.
                                                ;The offset and
                                                ;segment are stored
                                                ;in location 100-103H
0119                   start1:
0119    B8 0000          mov ax,0
011C    8E D8            mov ds,ax              ;data segment to 0000
011E    BB 0100          mov bx,0100H           ;pointer to int vecto
                                                ;100H
0121    83 7F 02 0       cmp word ptr [bx],0
0125    75 16            jne quit               ;program
                                                ;already run, exit
```

```
0127     83 3F 00      cmp word ptr2 [bx],0
012A     75 11         jne quit              ;program
                                             ;already run,
                                             ;exit
012C     B8 0103 R     mov ax,offset usrprg
012F     89 07         mov [bx],ax           ;program offset
0131     8C c8         mov ax,cs
0133     89 47 02      mov [bx+2],ax         ;data segment
0136     0E            push cs
0137     1F            pop ds
0138     BA 0141 R     mov dx,offset veryend
013B     CD 27         int 27h
013D                 quit:
013D     CD 20         int 20h
013F                 veryend:
013F                 double ends
                       end start
```

Example 2 places the assembly language subroutine in the specified area:

Example 2

```
10 I=0:JC=0
100 DIM A%(23)
150 MEM%=VARPTR(A%(1))
200 FOR I=1 TO 23
300 READ JC
400 POKE MEM%,JC
450 MEM%=MEM%+1
500 NEXT
600 C1%=2:C2%=3:C3%=0
700 TWOSUM=VARPTR(A%(1))
800 CALL TWOSUM(C1%,C2%,C3%)
900 PRINT C3%
950 END
1000 DATA &H55,&H8b,&Hec &H8b,&H76,&H08,&H8b,&H04,&H8b,&H76
1100 DATA &H0a,&H03,&H04,&H8b,&H7e,&H06,&H89,&H05,&H5d
1200 DATA &Hca,&H06,&H00
```

Appendix E Converting BASIC Programs to GW-BASIC

Programs written in a BASIC language other than GW-BASIC may require some minor adjustments before they can be run. The following sections describe these adjustments.

E.1 String Dimensions

Delete all statements used to declare the length of strings. A statement such as the following:

```
DIM A$(I,J)
```

which dimensions a string array for J elements of length I, should be converted to the following statement:

```
DIM A$(J)
```

Some GW-BASIC languages use a comma or ampersand (&) for string concatenation. Each of these must be changed to a plus sign (+), which is the operator for GW-BASIC string concatenation.

In GW-BASIC, the MID$, RIGHT$, and LEFT$ functions are used to take substrings of strings. Forms such as A$(I) to access the Ith character in A$, or A$(I,J) to take a substring of A$ from position I to position J, must be changed as follows:

Other BASIC:	**GW-BASIC:**
X$=A$(I)	X$=MID$(A$,I,1)
X$=A$(I,J)	X$=MID$(A$,I,J-I+1)

If the substring reference is on the left side of an assignment, and X$ is used to replace characters in A$, convert as follows:

Other BASIC:	GW-BASIC:
A$(I)=X$	MID$(A$,I,1)=X$
A$(I,J)=X$	MID$(A$,I,J-I+1)=X$

E.2 Multiple Assignments

Some GW-BASIC languages allow statements of the following form to set B and C equal to zero:

```
10 LET B=C=0
```

GW-BASIC would interpret the second equal sign as a logical operator and set B equal to −1 if C equaled 0. Convert this statement to two assignment statements:

```
10 C=0:B=0
```

E.3 Multiple Statements

Some GW-BASIC languages use a backslash (\) to separate multiple statements on a line. With GW-BASIC, be sure all elements on a line are separated by a colon (:).

E.4 MAT Functions

Programs using the MAT functions available in some GW-BASIC languages must be rewritten using FOR-NEXT loops to execute properly.

E.5 FOR-NEXT Loops

Some GW-BASIC languages will always execute a FOR-NEXT loop once, regardless of the limits. GW-BASIC checks the limits first and does not execute the loop if past limits.

Appendix F Communications

This appendix describes the GW-BASIC statements necessary to support RS-232 asynchronous communications with other computers and peripheral devices.

F.1 Opening Communications Files

The OPEN COM statement allocates a buffer for input and output in the same manner as the OPEN statement opens disk files.

F.2 Communications I/O

Since the communications port is opened as a file, all I/O statements valid for disk files are valid for COM.

COM sequential input statements are the same as those for disk files:

INPUT#
LINE INPUT#
INPUT$

COM sequential output statements are the same as those for diskette:

PRINT#
PRINT# USING

See the *GW-BASIC User's Reference* for more information on these statements.

F.3 The COM I/O Functions

The most difficult aspect of asynchronous communications is processing characters as quickly as they are received. At rates above 2400 baud (bps), it is necessary to suspend character transmission from the host long enough for the receiver to catch up. This can be done by sending XOFF (CTRL-S) to the host to temporarily suspend transmission, and XON (CTRL-Q) to resume, if the application supports it.

GW-BASIC provides three functions which help to determine when an overrun condition is imminent:

LOC(x)	Returns the number of characters in the input queue waiting to be read. The input queue can hold more than 255 characters (determined by the **/c:** switch). If there are more than 255 characters in the queue, LOC(x) returns 255. Since a string is limited to 255 characters, this practical limit alleviates the need for the programmer to test for string size before reading data into it.
LOF(x)	Returns the amount of free space in the input queue; that is **/c:**(*size*)-*number of characters in the input queue* LOF may be used to detect when the input queue is reaching storage capacity.
EOF(x)	True (−1), indicates that the input queue is empty. False (0) is returned if any characters are waiting to be read.

F.4 Possible Errors:

A "Communications buffer overflow" error occurs if a read is attempted after the input queue is full (that is, LOC(x) returns 0).

A "Device I/O" error occurs if any of the following line conditions are detected on receive: overrun error (OE), framing error (FE), or break interrupt (BI). The error is reset by subsequent inputs, but the character causing the error is lost.

A “Device fault” error occurs if data set ready (DSR) is lost during I/O.

A “Parity error” occurs if the PE (parity enable) option was used in the OPEN COM statement and incorrect parity was received.

F.5 The INPUT$ Function

The INPUT$ function is preferred over the INPUT and LINE INPUT statements for reading COM files, because all ASCII characters may be significant in communications. INPUT is least desirable because input stops when a comma or an enter is seen. LINE INPUT terminates when an enter is seen.

INPUT$ allows all characters read to be assigned to a string.

INPUT$ returns x characters from the y file. The following statements then are most efficient for reading a COM file:

```
10 WHILE NOT EOF(1)
20 A$=INPUT$(LOC(1),#1)
30 ...
40 ... Process data returned in A$ ...
50 ...
60 WEND
```

This sequence of statements translates: As long as something is in the input queue, return the number of characters in the queue and store them in A$. If there are more than 255 characters, only 255 are returned at a time to prevent string overflow. If this is the case, EOF(1) is false, and input continues until the input queue is empty.

GET and PUT Statements for COM Files

Purpose:

To allow fixed-length I/O for COM.

Syntax:

GET *filenumber*, *nbytes* **PUT** *filenumber*, *nbytes*

Comments:

filenumber is an integer expression returning a valid file number.

nbytes is an integer expression returning the number of bytes to be transferred into or out of the file buffer. *nbytes* cannot exceed the value set by the **/s:** switch when GW-BASIC was invoked.

Because of the low performance associated with telephone line communications, it is recommended that GET and PUT not be used in such applications.

Example:

The following TTY sample program is an exercise in communications I/O. It is designed to enable your computer to be used as a conventional terminal. Besides full-duplex communications with a host, the TTY program allows data to be downloaded to a file. Conversely, a file may be uploaded (transmitted) to another machine.

In addition to demonstrating the elements of asynchronous communications, this program is useful for transferring GW-BASIC programs and data to and from a computer.

Note

This program is set up to communicate with a DEC® SYSTEM-20 especially in the use of XON and XOFF. It may require modification to communicate with other types of hardware.

F.6 The TTY Sample Program

```
10 SCREEN 0,0:WIDTH 80
15 KEY OFF:CLS:CLOSE
20 DEFINT A-Z
25 LOCATE 25,1
30 PRINT STRING$(60," ")
40 FALSE=0:TRUE=NOT FALSE
50 MENU=5 'Value of MENU Key (^E)
60 XOFF$=CHR$(19):XON$=CHR$(17)
100 LOCATE 25,1:PRINT "Async TTY Program";
110 LOCATE 1,1:LINE INPUT "Speed?";"SPEED$
120 COMFIL$="COM1:,+SPEED$+",E,7"
130 OPEN COMFIL$ AS #1
140 OPEN "SCRN:"FOR OUTPUT AS #3
200 PAUSE=FALSE
210 A$=INKEY$:IF A$=""THEN 230
220 IF ASC(A$)=MENU THEN 300 ELSE PRINT #1,A$;
230 IF EOF(1) THEN 210
240 IF LOC(1)>128 THEN PAUSE=TRUE:PRINT #1,XOFF$;
250 A$=INPUT$(LOC(1),#1)
260 PRINT #3,A$;:IF LOC(1)>0 THEN 240
270 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
280 GOTO 210
300 LOCATE 1,1:PRINT STRING$(30,32):LOCATE 1,1
310 LINE INPUT "FILE?";DSKFIL$
400 LOCATE 1,1:PRINT STRING$(30,32):LOCATE 1,1
410 LINE INPUT"(T)ransmit or (R)eceive?";TXRX$
420 IF TXRX$="T" THEN OPEN DSKFIL$ FOR INPUT AS #2:GOTO 1000
430 OPEN DSKFIL$ FOR OUTPUT AS #2
440 PRINT #1,CHR$(13);
500 IF EOF(1) THEN GOSUB 600
510 IF LOC(1)>128 THEN PAUSE=TRUE:PRINT #1,XOFF$;
520 A$=INPUT$(LOC(1),#1)
530 PRINT #2,A$;:IF LOC(1)>0 THEN 510
540 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
550 GOTO 500
600 FOR I=1 TO 5000
610 IF NOT EOF(1) THEN I=9999
620 NEXT I
630 IF I>9999 THEN RETURN
640 CLOSE #2;CLS:LOCATE 25,10:PRINT "* Download complete *";
650 RETURN 200
1000 WHILE NOT EOF(2)
1010 A$=INPUT$(1,#2)
1020 PRINT #1,A$;
1030 WEND
1040 PRINT #1,CHR$(28);^Z to make close file.
```

```
1050 CLOSE #2:CLS:LOCATE 25,10:PRINT "** Upload complete **";
1060 GOTO 200
9999 CLOSE:KEY ON
```

F.7 Notes on the TTY Sample Program

Note

Asynchronous implies character I/O as opposed to line or block I/O. Therefore, all prints (either to the COM file or screen) are terminated with a semicolon (;). This retards the return line feed normally issued at the end of the PRINT statement.

Line Number	Comments
10	Sets the SCREEN to black and white alpha mode and sets the width to 80.
15	Turns off the soft key display, clears the screen, and makes sure that all files are closed.
20	Defines all numeric variables as integer, primarily for the benefit of the subroutine at 600-620. Any program looking for speed optimization should use integer counters in loops where possible.
40	Defines boolean true and false.
50	Defines the ASCII (ASC) value of the MENU key.
60	Defines the ASCII XON and XOFF characters.
100-130	Prints program ID and asks for baud rate (speed). Opens communications to file number 1, even parity, 7 data bits.
200-280	This section performs full-duplex I/O between the video screen and the device connected to the RS-232 connector as follows:

1. Read a character from the keyboard into A$. INKEY$ returns a null string if no character is waiting.
2. If a keyboard character is available, waiting, then:

   If the character is the MENU key, the operator is ready to down-load a file. Get filename.

   If the character (A$) is not the MENU key, send it by writing to the communications file (PRINT #1...).
3. If no character is waiting, check to see if any characters are being received.
4. At 230, see if any characters are waiting in COM buffer. If not, go back and check the keyboard.
5. At 240, if more than 128 characters are waiting, set PAUSE flag to indicate that input is being suspended. Send XOFF to host, stopping further transmission.
6. At 250-260, read and display contents of COM buffer on screen until empty. Continue to monitor size of COM buffer (in 240). Suspend transmission if reception falls behind.
7. Resume host transmission by sending XON only if suspended by previous XOFF.
8. Repeat process until the MENU key is pressed.

300-320	Get disk filename to be down-loaded to. Open the file as number 2.
400-420	Asks if file named is to be transmitted (up-loaded) or received (down-loaded).
430	Receive routine. Sends a RETURN to the host to begin the down-load. This program assumes that the last command sent to the host was to begin such a transfer and was missing only the terminating return. If a DEC system is the host, such a command might be

```
COPY TTY:=MANUAL.MEM (MENU Key)
```

if the MENU key was struck instead of RETURN.

500	When no more characters are being received, (LOC(x) returns 0), the program performs a timeout routine.
510	If more than 128 characters are waiting, signal a pause and send XOFF to the host.
520-530	Read all characters in COM queue (LOC(x)) and write them to diskette (PRINT #2...) until reception is caught up to transmission.
540-550	If a pause is issued, restart host by sending XON and clearing the pause flag. Continue the process until no characters are received for a predetermined time.
600-650	Time-out subroutine. The FOR loop count was determined by experimentation. If no character is received from the host for 17-20 seconds, transmission is assumed complete. If any character is received during this time (line 610), then set *n* well above the FOR loop range to exit loop and return to caller. If host transmission is complete, close the disk file and resume regular activities.
1000-1060	Transmit routine. Until end of disk file, read one character into A$ with INPUT$ statement. Send character to COM device in 1020. Send a ^Z at end of file in 1040 in case receiving device needs one to close its file. Lines 1050 and 1060 close disk file, print completion message, and go back to conversation mode in line 200.
9999	Presently not executed. As an exercise, add some lines to the routine 400-420 to exit the program via line 9999. This line closes the COM file left open and restores the function key display.

Appendix G Hexadecimal Equivalents

Table G.1 lists decimal and binary equivalents to hexadecimal values.

Table G.1

Decimal and Binary Equivalents to Hexadecimal Values

Hexadecimal Value	Equals Decimal:	Equals Binary:
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Table G.2 lists decimal equivalents to hexadecimal values.

Table G.2

Decimal Equivalents to Hexadecimal Values

Hexadecimal Value	Equals Decimal:	Hexadecimal Value:	Equals Decimal:
0	0	80	128
1	1	.	
2	2	.	
3	3	.	
4	4	90	144
5	5	.	
6	6	.	
7	7	.	
8	8	A0	160
9	9	.	
A	10	.	
B	11	.	
C	12	B0	176
D	13	.	
E	14	.	
F	15	.	
10	16	C0	192
11	17	.	
12	18	.	
13	19	.	
14	20	D0	208
15	21	.	
16	22	.	
17	23	.	
18	24	E0	224
19	25	.	
1A	26	.	
1B	27	.	
1C	28	F0	240
1D	29	100	256
1E	30	200	512
1F	31	300	768
20	32	400	1024
.	.	500	1280
.	.	600	1536
.	.	700	1792

Table G.2 *(continued)*

Hexadecimal Value	Equals Decimal:	Hexadecimal Value:	Equals Decimal:
30	48	800	2048
.	.	900	2304
.	.	A00	2560
.	.	B00	2816
40	64	C00	3072
.	.	D00	3328
.	.	E00	3584
.	.	F00	3840
50	80	1000	4096
.	.	2000	8192
.	.	3000	12288
.	.	4000	16384
60	96	5000	20480
.	.	6000	24576
.	.	7000	28672
.	.	8000	32768
70	112	9000	36864
.	.	A000	40960
.	.	B000	45056
.	.	C000	49152
		D000	53248
		E000	57344
		F000	61440

Appendix H
Key Scan Codes

Keytop Legend	**Scancode**
ESC	01
1/!	02
2/@	03
3/#	04
	05
5/%	06
6/^	07
7/&	08
8/*	09
9/(	0A
0/)	0B
-/_	0C
=/+	0D
BACKSPACE	0E
TAB	0F
Q	10
W	11
E	12
R	13
T	14
Y	15
U	16
I	17
O	18
P	19
[/{	1A
]/}	1B
ENTER	1C
CTRL	1D
A	1E
S	1F
D	20
F	21
G	22
H	23
J	24

Keytop Legend	Scancode
K	25
L	26
;/:	27
’/"	28
’/˜	29
Left SHIFT	2A
/|	2B
Z	2C
X	2D
C	2E
V	2F
B	30
N	31
M	32
,/<	33
//?	35
Right SHIFT	36
*/PRTSC	37
ALT	38
SPACEBAR	39
CAPS LOCK	3A
F1	3B
F2	3C
F3	3D
F4	3E
F5	3F
F6	40
F7	41
F8	42
F9	43
F10	44
NUM LOCK	45
SCROLL LOCK	46
7/HOME	47
8/CURSOR UP	48
9/PGUP	49

Keytop Legend	Scancode
-	4A
4/CURSOR LEFT	4B
5	4C
6/CURSOR RIGHT	4D
+	4E
1/END	4F
2/CURSOR DOWN	50
3/PGDN	51
0/INS	52
./DEL	53

Appendix I
Characters Recognized by GW-BASIC

The GW-BASIC character set includes all characters that are legal in GW-BASIC commands, statements, functions, and variables. The set comprises alphabetic, numeric, and special characters.

The alphabetic characters in GW-BASIC are the uppercase and lowercase letters of the alphabet.

The numeric characters in GW-BASIC are the digits 0 through 9.

The following special characters and terminal keys are recognized by GW-BASIC:

Character	**Description**
	Blank.
=	Equal sign or assignment symbol.
+	Plus sign or string concatenation.
-	Minus sign.
*	Asterisk or multiplication symbol.
/	Slash or division symbol.
^	Caret, exponentiation symbol, or CTRL key.
(	Left parenthesis.
)	Right parenthesis.
%	Percent or integer declaration.
#	Number sign or double-precision declaration.
$	Dollar sign or string declaration.
!	Exclamation point or single-precision declaration.

[	Left bracket.
]	Right bracket.
,	Comma.
“”	Double quotation marks or string delimiter.
.	Period, dot, or decimal point.
’	Single quotation mark, apostrophe, or remark indicator.
;	Semicolon or carriage return suppressor.
:	Colon or line statement delimiter.
&	Ampersand or descriptor for hexadecimal and octal number conversion.
?	Question mark.
<	Less than symbol.
>	Greater than symbol.
\	Backslash or integer division symbol.
@	“At” sign.
_	Underscore.
BACKSPACE	Deletes last character typed.
ESC	Erases the current logical line from the screen.
TAB	Moves print position to next tab stop. Tab stops are every eight columns.
CURSOR	Moves cursor to next physical line.
RETURN	Terminates input to a line and moves cursor to beginning of the next line, or executes statement in direct mode.

Glossary

abend

An acronym for *abnormal end of task.* An abend is the termination of computer processing on a job or task prior to its completion because of an error condition that cannot be resolved by programmed recovery procedures.

access

The process of seeking, reading, or writing data on a storage unit.

access methods

Techniques and programs used to move data between main memory and input/output devices.

accuracy

The degree of freedom from error. Accuracy is often confused with *precision,* which refers to the degree of preciseness of a measurement.

acronym

A word formed by the initial letters of words or by initial letters plus parts of several words. Acronyms are widely used in computer technology. For example, *COBOL* is an acronym for COmmon Business Oriented Language.

active partition

A section of the computer's memory that houses the *operating system* being used.

address

A name, label, or number identifying a register, location or unit where information is stored.

algebraic language

A language whose statements are structured to resemble the structure of algebraic expression. Fortran is a good example of an algebraic language.

algorithm

A set of well-defined rules or procedures to be followed in order to obtain the solution of a problem in a finite number of steps. An algorithm can involve arithmetic, algebraic, logical and other types of procedures and instructions. An algorithm can be simple or complex. However, all algorithms must produce a solution within a finite number of steps. Algorithms are fundamental when using a computer to solve problems, because the computer must be supplied with a specific set of instructions that yields a solution in a reasonable length of time.

alphabetic

Data representation by alphabetical characters in contrast to numerical; the letters of the alphabet.

alphanumeric

A contraction of the words alphabetic and numeric; a set of characters including letters, numerals, and special symbols.

application

The system or problem to which a computer is applied. Reference is often made to an application as being either of the *computational type*, in which arithmetic computations predominate, or of the *data processing type*, in which data handling operations predominate.

application program

A computer program designed to meet specific user needs.

argument

1. A type of variable whose value is not a direct function of another variable. It can represent the location of a number in a mathematical operation, or the number with which a function works to produce its results.
2. A known reference factor that is required to find a desired item (function) in a table. For example, in the square root function SQRT(X), X is the argument. The value of X determines the square root value returned by this function.

array

1. An organized collection of data in which the argument is positioned before the function.
2. A group of items or elements in which the position of each item or element is significant. A multiplication table is a good example of an array.

ASCII

Acronym for American Standard Code for Information Interchange. ASCII is a standardized 8-bit code used by most computers for interfacing.

ASCII was developed by the American National Standards Institute (ANSI). It uses 7 binary bits for information and the 8th bit for parity purposes.

assembler

A computer program that produces a machine-language program which may then be directly executed by the computer.

assembly language

A symbolic language that is machine-oriented rather than problem-oriented. A program in an assembly language is converted by an assembler to a machine-language program. Symbols representing storage locations are converted to numerical storage locations; symbolic operation codes are converted to numeric operation codes.

asynchronous

1. Not having a regular time or clocked relationship. See *synchronous.*
2. A type of computer operation in which a new instruction is initiated when the former instruction is completed. Thus, there is no regular time schedule, or clock, with respect to instruction sequence. The current instruction must be complete before the next is begun, regardless of the length of time the current instruction takes.

asynchronous communication

A way of transmitting data serially from one device to another, in which each transmitted character is preceded by a start bit and followed by a stop bit. This is also called start/stop transmission.

back up

1. A second copy of data on a diskette or other medium, ensuring recovery from loss or destruction of the original media.
2. On-site or remote equipment available to complete an operation in the event of primary equipment failure.

BASIC

Acronym for Beginner's All-purpose Symbolic Instruction Code. BASIC is a computer programming language developed at Dartmouth College as an instructional tool in teaching fundamental programming concepts. This language has since gained wide acceptance as a time-sharing language and is considered one of the easiest programming languages to learn.

batch processing

A method of operating a computer so that a single program or set of related programs must be completed before the next type of program is begun.

baud

A unit of measurement of data processing speed. The speed in bauds is the number of signal elements per second. Since a signal element can represent more than one bit, baud is not synonymous with bits-per-second. Typical baud rates are 110, 300, 1200, 2400, 4800, and 9600.

binary

1. A characteristic or property involving a choice or condition in which there are two possibilities.
2. A numbering system which uses 2 as its base instead of 10 as in the decimal system. The binary system uses only two digits, 0 and 1, in its written form.
3. A device whose design uses only two possible states or levels to perform its functions. A computer executes programs in binary form.

binary digit

A quantity which is expressed in the binary digits of 0 and 1.

bit

A contraction of "binary digit". A bit can either be 0 or 1, and is the smallest unit of information recognizable by a computer.

block

An amount of storage space or data, of arbitrary length, usually contiguous, and often composed of several similar records, all of which are handled as a unit.

boolean logic

A field of mathematical analysis in which comparisons are made. A programmed instruction can cause a comparison of two fields of data, and modify one of those fields or another field as a result of comparison. This system was formulated by British mathematician George Boole (1815-1864). Some boolean operators are OR, AND, NOT, XOR, EQV, and IMP.

boot

A machine procedure that allows a system to begin operations at the desired level by means of its own initiation. The first few instructions are loaded into a computer from an input device. These instructions allow the rest of the system to be loaded. The word *boot* is abbreviated from the word *bootstrap.*

bps

Bits per second.

buffer

A temporary storage area from which data is transferred to or from various devices.

built-in clock

A real-time clock that lets your programs use the time of day and date. Built into MS-DOS, it lets you set the timing of a program. It can be used to keep a personal calendar, and it automatically measures elapsed time.

byte

An element of data which is composed of eight data bits plus a parity bit, and represents either one alphabetic or special character, two decimal digits, or eight binary bits. Byte is also used to refer to a sequence of eight binary digits handled as a unit. It is usually encoded in the *ASCII* format.

calculation

A series of numbers and mathematical signs that, when entered into a computer, is executed according to a series of instructions.

central processor (CPU)

The heart of the computer system, where data is manipulated and calculations are performed. The CPU contains a control unit to interpret and execute the program and an arithmetic-logic unit to perform computations and logical processes. It also routes information, controls input and output, and temporarily stores data.

chaining

The use of a pointer in a record to indicate the address of another record logically related to the first.

character

Any single letter of the alphabet, numeral, punctuation mark, or other symbol that a computer can read, write, and store. Character is synonymous with the term *byte*.

COBOL

Acronym for COmmon Business-Oriented Language, a computer language suitable for writing complicated business applications programs. It was developed by CODASYL, a committee representing the U. S. Department of Defense, certain computer manufacturers, and major users of data processing equipment. COBOL is designed to express data manipulations and processing problems in English narrative form, in a precise and standard manner.

code

1. To write instructions for a computer system
2. To classify data according to arbitrary tables
3. To use a machine language
4. To program

command

A pulse, signal, word, or series of letters that tells a computer to start, stop, or continue an operation in an instruction. Command is often used incorrectly as a synonym for *instruction*.

compatible

A description of data, programs or equipment that can be used between different kinds of computers or equipment.

compiler

A computer program that translates a program written in a problem-oriented language into a program of instructions similar to, or in, the language of the computer.

computer network

A geographically dispersed configuration of computer equipment connected by communication lines and capable of load sharing, distributive processing, and automatic communication between the computers within the network.

concatenate

To join together data sets, such as files, in a series to form one data set, such as one new file. The term concatenate literally means "to link together." A concatenated data set is a collection of logically connected data sets.

configuration

In hardware, a group of interrelated devices that constitute a system. In software, the total of the software modules and their interrelationships.

constant

A never-changing value or data item.

coprocessor

A microprocessor device connected to a central microprocessor that performs specialized computations (such as floating-point arithmetic) much more efficiently than the CPU alone.

cursor

A blinking line or box on a computer screen that indicates the next location for data entry.

data

A general term used to signify all the basic information elements that can be produced or processed by a computer. See *information.*

data element

The smallest named physical data unit.

data file

A collection of related data records organized in a specific manner. Data files contain computer records which contain information, as opposed to containing data handling information or a program.

debug

The process of checking the logic of a computer program to isolate and remove mistakes from the program or other software.

default

An action or value that the computer automatically assumes, unless a different instruction or value is given.

delimit

To establish parameters; to set a minimum and a maximum.

delimiter

A character that marks the beginning or end of a unit of data on a storage medium. Commas, semi-colons, periods, and spaces are used as delimiters to separate and organize items of data.

detail file

A data file composed of records having similar characteristics, but containing data which is relatively changeable by nature, such as employee weekly payroll data. Compare to *master file.*

device

A piece of hardware that can perform a specific function. A printer is an example of a device.

diagnostic programs

Special programs used to align equipment or isolate equipment malfunctions.

directory

A table that gives the name, location, size, and the creation or last revision date for each file on the storage media.

diskette

A flat, flexible platter coated with magnetic material, enclosed in a protective envelope, and used for storage of software and data.

Disk Operating System

A collection of procedures and techniques that enable the computer to operate using a disk drive system for data entry and storage. Disk Operating System is usually abbreviated to *DOS*.

DOS

The acronym for Disk Operating System. DOS rhymes with "boss."

double-density

A type of diskette that has twice the storage capacity of standard single-density diskettes.

double-precision

The use of two computer words to represent each number. This technique allows the use of twice as many digits as are normally available and is used when extra precision is needed in calculations.

double-sided

A term that refers to a diskette that can contain data on both surfaces of the diskette.

drive

A device that holds and manipulates magnetic media so that the CPU can read data from or write data to them.

end-of-file mark (EOF)

A symbol or machine equivalent that indicates that the last record of a file has been read.

erase

To remove or replace magnetized spots from a storage medium.

error message

An audible or visual indication of hardware or software malfunction or of an illegal data-entry attempt.

execute

To carry out an instruction or perform a routine.

exponent

A symbol written above a factor and on the right, telling how many times the factor is repeated. In the example of A^2, A is the factor and 2 is the exponent. A^2 means A times A ($A \times A$).

extension

A one-to-three-character set that follows a filename. The extension further defines or clarifies the filename. It is separated from the filename by a period(.).

field

An area of a record that is allocated for a specific category of data.

file

A collection of related data or programs that is treated as a unit by the computer.

file protection

The devices or procedures that prevent unintentional erasure of data on a storage device, such as a diskette.

file structure

A conceptual representation of how data values, records, and files are related to each other. The structure usually implies how the data is stored and how the data must be processed.

filename

The unique name, usually assigned by a user, that identifies one file for all subsequent operations that use that file.

fixed disk

A hard disk enclosed in a permanently-sealed housing that protects it from environmental interference. Used for storage of data.

floating-point arithmetic

A method of calculation in which the computer or program automatically records, and accounts for, the location of the radix point. The programmer need not consider the radix location.

floating-point routine

A set of program instructions that permits a floating-point mathematics operation in a computer which lacks the feature of automatically accounting for the radix point.

format

A predetermined arrangement of data that structures the storage of information on an external storage device.

function

A computer action, as defined by a specific instruction. Some GW-BASIC functions are COS, EOF, INSTR, LEFT$, and TAN.

function keys

Specific keys on the keyboard that, when pressed, instruct the computer to perform a particular operation. The function of the keys is determined by the applications program being used.

GIGO

An informal term that indicates sloppy data processing; an acronym for Garbage In Garbage Out. The term GIGO is normally used to make the point that if the input data is bad (garbage in) then the output data will also be bad (garbage out).

global search

Used in reference to a variable (character or command), a global search causes the computer to locate all occurrences of that variable.

graphics

A hardware/software capability to display objects in pictures, rather than words, usually on a graphic (CRT) display terminal with line-drawing capability and permitting interaction, such as the use of a light pen.

hard copy

A printed copy of computer output in a readable form, such as reports, checks, or plotted graphs.

hardware

The physical equipment that comprises a system.

hexadecimal

A number system with a base, or radix, of 16. The symbols used in this system are the decimal digits 0 through 9 and six additional digits which are generally represented as A, B, C, D, E, and F.

hidden files

Files that cannot be seen during normal directory searches.

hierarchical directories

See *tree-structured directories*.

housekeeping functions

Routine operations that must be performed before the actual processing begins or after it is complete.

information

Facts and knowledge derived from data. The computer operates on and generates data. The meaning derived from the data is information; that is, information results from data. The two words are not synonymous, although they are often used interchangeably.

interpreter

A program that reads, translates and executes a user's program, such as one written in the BASIC language, one line at a time. A compiler, on the other hand, reads and translates the entire user's program before executing it.

input

1. The process or device concerning the entry of data into a computer.
2. Actual data being entered into a computer.

input/output

A general term for devices that communicate with a computer. Input/output is usually abbreviated as I/O.

instruction

A program step that tells the computer what to do next. Instruction is often used incorrectly as a synonym for command.

integer

A complete entity, having no fractional part. The whole or natural number. For example, 65 is an integer; 65.1 is not.

integrated circuit

A complete electronic circuit contained in a small semiconductor component.

interface

An information interchange path that allows parts of a computer, computers, and external equipment (such as printers, monitors, or modems), or two or more computers to communicate or interact.

I/O

The acronym for input/output.

job

A collection of tasks viewed by the computer as a unit.

K

The symbol signifying the quantity 2^{10}, which is equal to 1024. K is sometimes confused with the symbol *k* (kilo), which is equal to 1000.

logarithm

A logarithm of a given number is the value of the exponent indicating the power required to raise a specified constant, known as the base, to produce that given number. That is, if B is the base, N is the given number and L is the logarithm, then BL = N. Since 10^3 = 1000, the logarithm to the base 10 of 1000 is 3.

loop

A series of computer instructions that are executed repeatedly until a desired result is obtained or a predetermined condition is met. The ability to loop and reuse instructions eliminates countless repetitious instructions and is one of the most important attributes of stored programs.

M

The symbol signifying the quantity 1,000,000 (10^6). When used to denote storage, it more precisely refers to 1,048,576 (2^{20}).

mantissa

The fractional or decimal part of a logarithm of a number. For example, the logarithm of 163 is 2.212. The mantissa is 0.212, and the characteristic is 2.0.

In floating-point numbers, the mantissa is the number part. For example, the number 24 can be written as 24,2 where 24 is the mantissa and 2 is the exponent. The floating-point number is read as $.24 \times 10^2$, or 2^4.

master file

A data file composed of records having similar characteristics that rarely change. A good example of a master file would be an employee name and address file that also contains social security numbers and hiring dates.

media

The plural of medium.

medium

The physical material on which data is recorded and stored. Magnetic tape, punched cards, and diskettes are examples of media.

memory

The high-speed work area in the computer where data can be held, copied, and retrieved.

menu

A list of choices from which an operator can select a task or operation to be performed by the computer.

microprocessor

A semiconductor *central processing unit* (CPU) in a computer.

modem

Acronym for modulator demodulator. A modem converts data from a computer to analog signals that can be transmitted through telephone lines, or converts the signals from telephone lines into a form the computer can use.

MS-DOS

Acronym for Microsoft Disk Operating System.

nested programs or subroutines

A program or subroutine that is incorporated into a larger routine to permit ready execution or access of each level of the routine. For example, nesting loops involves incorporating one loop of instructions into another loop.

null

Empty or having no members. This is in contrast to a blank or zero, which indicates the presence of no information. For example, in the number 540, zero contains needed information.

numeric

A reference to numerals as opposed to letters or other symbols.

octal number system

A representation of values or quantities with octal numbers. The octal number system uses eight digits: 0, 1, 2, 3, 4, 5, 6, and 7, with each position in an octal numeral representing a power of 8. The octal system is used in computing as a simple means of expressing binary quantities.

operand

A quantity or data item involved in an operation. An operand is usually designated by the address portion of an instruction, but it may also be a result, a parameter, or an indication of the name or location of the next instruction to be executed.

operating system

An organized group of computer *instructions* that manage the overall operation of the computer.

operator

A symbol indicating an operation and itself the subject of the operation. It indicates the process that is being performed. For example, + is addition, − is subtraction, × is multiplication, and / is division.

option

An add-on device that expands a system's capabilities.

output

Computer results, or data that has been processed.

parallel output

The method by which all bits of a binary word are transmitted simultaneously.

parameter

A *variable* that is given a value for a specific program or run. A definable characteristic of an item, device, or system.

parity

An extra-bit of code that is used to detect data errors in memory by making the sum of the active bit in a data word either an odd or an even number.

partition

An area on a *fixed disk* set aside for a specific purpose, such as a location for an operating system.

peripheral

An external input/output, or storage device.

pixel

The acronym for picture element. A pixel is a single dot on a monitor that can be addressed by a single bit.

port

The entry channel to and from the central computer for connection of a communications line or other *peripheral* device.

power

The functional area of a system that transforms an external power source into internal DC supply voltage.

program

A series of instructions or statements in a form acceptable to a computer, designed to cause the computer to execute a series of operations. Computer programs include software such as operating systems, assemblers, compilers, interpreters, data management systems, utility programs, sort-merge programs, and maintenance/diagnostic programs, as well as application programs such as payroll, inventory control, and engineering analysis programs.

prompt

A character or series of characters that appear on the screen to request input from the user.

RAM

Acronym for *random-access memory.*

radian

The natural unit of measure of the angle between two intersecting half-lines on the angles from one half-line to another intersecting half-line. It is the angle subtended by an arc of a circle equal in length to the radius of the circle. As the circumference of a circle is equal to 2π times its radius, the number of radians in an angle of 360° or in a complete turn is 2π.

radix

A number that is arbitrarily made the fundamental number of a system of numbers; a base. Thus, 10 is the radix, or base, of the common system of logarithms, and also of the decimal system of enumeration.

random-access memory

The system's high-speed work area that provides access to memory storage locations by using a system of vertical and horizontal coordinates. The computer can write information into or read information from the random-access memory. Random-access memory is often called *RAM*.

raster unit

On a graphic display screen, a raster unit is the horizontal or vertical distance between two adjacent addressable points on the screen.

read-only memory

A type of memory that contains permanent data or instructions. The computer can read from but not write to the read-only memory. Read-only memory is often called *ROM*.

real number

An ordinary number, either rational or irrational; a number in which there is no imaginary part, a number generated from the single unit, 1; any point in a continuum of natural numbers filled in with all rationals and all irrationals and extended indefinitely, both positive and negative.

real time

1. The actual time required to solve a problem.
2. The process of solving a problem during the actual time that a related physical process takes place so that results can be used to guide the physical process.

remote

A term used to refer to devices that are located at sites away from the central computer.

reverse video

A display of characters on a background, opposite of the usual display.

ROM

Acronym for *read-only memory*.

RS-232

A standard communications interface between a *modem* and terminal devices that complies with EIA Standard RS-232.

serial output

Sending only one bit at a time to and from interconnected devices.

single-density

The standard recording density of a diskette. Single-density diskettes can store approximately 3400 bits per inch (bpi).

single-precision value

The number of words or storage positions used to denote a number in a computer. Single-precision arithmetic is the use of one word per number, double-precision arithmetic is the use of two words per number, and so on. For variable word-length computers, precision is the number of digits used to denote a number. The higher the precision, the greater the number of decimal places that can be carried.

single-sided

A term used to describe a diskette that contains data on one side only.

software

A string of instructions that, when executed, direct the computer to perform certain functions.

stack architecture

An architecture wherein any portion of the external memory can be used as a last-in, first-out stack to store/retrieve the contents of the accumulator, the flags, or any of the data registers. Many units contain a 16-bit stack pointer to control the addressing of this external stack. One of the major advantages of the stack is that multiple-level interrupts can be handled easily, since complete system status can be saved when an interrupt occurs and then be restored after the interrupt. Another major advantage is that almost unlimited subroutine nesting is possible.

statement

A high-level language instruction to the computer to perform some sequence of operations.

synchronous

A type of computer operation in which the execution of each instruction or each event is controlled by a clock signal: evenly spaced pulses that enable the logic gates for the execution of each logic step. A synchronous operation can cause time delays by causing waiting for clock signals although all other signals at a particular logic gate were available. See *asynchronous.*

switch

An instruction, added to a command, that designates a course of action, other than default, for the command process to follow.

syntax

Rules of statement structure in a programming language.

system

A collection of hardware, software, and firmware that is interconnected to operate as a unit.

task

A machine run; a program in execution.

toggle

Alternation of function between two stable states.

track

A specific area on a moving-storage medium, such as a diskette, disk, or tape cartridge, that can be accessed by the drive heads.

tree-structured directory

A file-organization structure, consisting of directories and subdirectories that, when diagrammed, resembles a tree.

truncation

To end a computation according to a specified rule; for example, to drop numbers at the end of a line instead of rounding them off, or to drop characters at the end of a line when a file is copied.

upgrade

To expand a system by installing options or using revised software.

utility function

Computer programs, dedicated to one particular task, that are helpful in using the computer. For example, FDISK, for setting up partitions on the fixed disk.

variable

A quantity that can assume any of a set of values as a result of processing data.

volume label

The name for the contents of a diskette or a partition on a fixed disk.

word

The set of bits comprising the largest unit that the computer can handle in a single operation.

write-protect notch

A cut-out opening in the sealed envelope of a diskette that, when covered, prevents writing or adding text to the diskette, but allows information to be read from the diskette.

Index