

INSIDE FRONT COVER - BLANK

TITLE PAGE

1st degree headlines

Beginning of general copy  
(bold weight)

Beginning of specific copy  
(regular weight)

TI-99/8, Book 2, General Information, Part 1

1523Pa001 on ti85

Cover Copy:

Programmer's Guide  
for the Computer 99/8

Texas Instruments  
Computer 99/8

[GRAPHIC OR PHOTO OF  
PRODUCT]

Programmer's Guide  
for the Computer 99/8

Copyright © 1981 by Texas Instruments  
Incorporated.

Folio

## Table of Contents

General Information on TI Extended BASIC II	11
Features of TI Extended BASIC II	11
Differences between TI Extended BASIC II and Previous Versions of TI BASIC	11
Modes in TI Extended BASIC II	11
Error Correction and Editing	11
Edit Mode	11
Special Editing Keys	11
Overview of TI Extended BASIC II	11
Commands	11
Assignments and Input	11
Output	11
Functions, Subroutines, and Subprograms	11
Sound, Speech, and Color	11
Sprites	11
Debugging	11
Error Handling	11
Pre-scan—!@P— and !@P+	11
TI Extended BASIC II Conventions	11
Running a Program on Powerup	XX
Files	XX
Line Numbers	XX
Lines	XX
Special Symbols	XX
Spaces	XX
Numeric Data	XX
Numeric Constants	XX
String Constants	XX
Variables	XX
Numeric Expressions	XX
String Expressions	XX
Relational Expressions	XX
Logical Expressions	XX
Graphics Modes	XX
Margins and the Screen Window	XX
Defaults	XX

File Commands	XX
File Specifications	XX
The HEX-BUS™ Interface	XX
Reference Guide for TI Extended BASIC II	XX
Notational Conventions for Format Lines	XX
General Descriptions	XX
ABS	XX
ACCEPT	XX
ASC	XX
ATN	XX
BREAK	YY
BYE	XX
CALL	XX
CHAR	XX
CHARPAT	XX
CHARSET	XX
CHRS	XX
CLEAR	XX
CLOSE	XX
COINC	XX
COLOR	XX
CONTINUE	XX
COS	XX
DATA	XX
DCOLOR	XX
DEF	XX
DELETE	XX
DELSPRITE	XX
DIM	XX
DISPLAY	XX
DISPLAY USING	XX
DISTANCE	XX
DRAW	XX
DRAWTO	XX
END	XX
EOF	XX
ERR	XX

1st degree headlines

Beginning of general copy  
(bold weight)

EXP .....XX  
 FILL .....XX  
 FOR TO (STEP) .....XX  
 FREESPACE(0) .....XX  
 GCHAR .....XX  
 GOSUB .....XX  
 GOTO .....XX  
 GRAPHICS .....XX  
 HCHAR .....XX  
 IF THEN [ELSE] .....XX  
 IMAGE .....XX  
 INIT .....XX  
 INPUT .....XX  
 INT .....XX  
 INTEGER .....XX  
 JOYST .....XX  
 KEY .....XX  
 LEN .....XX  
 LET .....XX  
 LINK .....XX  
 LINPUT .....XX  
 LIST .....XX  
 LOAD .....XX  
 LOCATE .....XX  
 LOG .....XX  
 MAGNIFY .....XX  
 MARGINS .....XX  
 MAX .....XX  
 MERGE .....XX  
 MIN .....XX  
 MOTION .....XX  
 NEW .....XX  
 NEXT .....XX  
 NUMBER .....XX  
 OLD .....XX  
 ON BREAK .....XX  
 ON ERROR .....XX  
 ON COSUB .....XX

ON GOTO .....XX  
 ON WARNING .....XX  
 OPEN .....XX  
 OPTION BASE .....XX  
 PATTERN .....XX  
 PEEK .....XX  
 PEEKV .....XX  
 PI .....XX  
 POKEV .....XX  
 POS .....XX  
 POSITION .....XX  
 PRINT .....XX  
 PRINT USING .....XX  
 RANDOMIZE .....XX  
 READ .....XX  
 REAL .....XX  
 REC .....XX  
 REM .....XX  
 RESEQUENCE .....XX  
 RESTORE .....XX  
 RETURN .....XX  
 RND .....XX  
 RPT\$ .....XX  
 RUN .....XX  
 SAVE .....XX  
 SAY .....XX  
 SCREEN .....XX  
 SEX\$ .....XX  
 SGN .....XX  
 SIN .....XX  
 SOUND .....XX  
 SPGET .....XX  
 SPRITE .....XX  
 SQR .....XX  
 STOP .....XX  
 STR\$ .....XX  
 SUB .....XX  
 SUBEND .....XX

Philo

4

5

1st degree headlines

Beginning of general copy  
(bold weight)

SUBEXIT .....	XX
TAB .....	XX
TAN .....	XX
TERMCHAR .....	XX
TRACE .....	XX
UNBREAK .....	XX
UNTRACE .....	XX
VAL .....	XX
VALHEX .....	XX
VCHAR .....	XX
VERSION .....	XX
The p System .....	XX
Loading an Existing Program .....	XX
Using the System Commands .....	XX
Prompts .....	XX
Prompts for Filenames .....	XX
p-System Commands .....	XX
A(ssem) .....	XX
Comp .....	XX
E(dit) .....	XX
F(ile) .....	XX
G(o) .....	XX
H(alt) .....	XX
I(nitalize) .....	XX
L(ink) .....	XX
M(onitor Options) .....	XX
R(un) .....	XX
U(ser Restart) .....	XX
X(ecute) .....	XX
Execution Option Strings .....	XX
Alternate Prefixes and Libraries .....	XX
Redirection .....	XX
Error Messages .....	XX

Appendices .....	XX
Appendix A—ASCII Character Codes .....	XX
Appendix B—Function Key Codes .....	XX
Appendix C—Control Key Codes .....	XX
Appendix D—Keyboard Mapping .....	XX
Appendix E—Character Sets .....	XX
Appendix F—Accuracy Information .....	XX
Appendix G—Reserved Words .....	XX
Appendix H—Musical Tone Frequencies .....	XX
Appendix I—Trigonometric Calculations and Restrictions .....	XX
Appendix J—Color Codes .....	XX
Appendix K—Color Combinations .....	XX
Appendix L—List of Speech Words .....	XX
Appendix M—Adding Suffixes to Speech Words .....	XX
Appendix N—Pattern-Identifier Conversion Table .....	XX
Appendix O—Assembly-Language Support Routines .....	XX
Appendix P—Assembly-Language Error Codes .....	XX
Appendix Q—Allowable Peripheral Commands .....	XX
Appendix R—Transfer Raw Data .....	XX
Appendix S—Using the Computer 99/8 as a Slave Device .....	XX
Appendix T—Input/Output Opcodes .....	XX
Appendix U—I/O Error Messages .....	XX
Appendix V—HEX-BUS™ Error Messages .....	XX
Appendix W—TI Extended BASIC II Error Messages .....	XX
Glossary .....	XX
Index .....	XX

## Features of TI Extended BASIC II

Texas Instruments Extended BASIC II is a powerful computer programming language that is built into the Texas Instruments Computer 99/8. It has the features expected from a high-level language plus additional features not available in many other languages.

TI Extended BASIC II enhances the capability and flexibility of your computer system with these features:

• **Input and Output**—The ACCEPT statement allows the input of data from anywhere on the screen. With this statement you can clear the screen, accept only certain characters, and limit the number of characters entered. The DISPLAY statement enables you to put data anywhere on the screen, and DISPLAY USING, PRINT USING, and IMAGE are included for ease in formatting data on the display screen and for peripheral devices.

• **Subprograms**—Subprograms with local variables (affecting only values within the subprogram) can be written in TI Extended BASIC II. Commonly used subprograms may be stored on a diskette and added to programs as needed. Statements included are SUB, SUBEND, and SUBEXIT. The MERGE and SAVE commands enable you to merge programs and subprograms from diskettes.

• **Sprites**—Sprites are specially defined graphics with the ability to move smoothly on the screen. To support the sprite capability, the following subprograms are included in TI Extended BASIC II: COINC, DELSPRITE, DISTANCE, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION, and

SPRITE. The COLOR and CHAR subprograms can also affect sprites.

• **Data Types**—TI Extended BASIC II allows both integer and real data types for numeric variables. Because the integer data type uses less storage space and requires less processing time, using this data type when applicable (for example, with FOR-NEXT loops) facilitates faster program execution.

• **Six Different Graphics Modes**—A wide variety of display techniques and applications, including dividing graphics and text, are available with the six graphics modes:

**Pattern Mode**—The default mode used when TI Extended BASIC II is selected. Characters are in 24 rows and 32 columns, with each character defined by an 8 X 8 pixel configuration.

**Text Mode**—Characters are in 24 rows and 40 columns, with each character defined by an 8 X 6 pixel configuration.

**Split-Screen Mode: Text-High**—The screen is split into two portions: the text portion (the top one-third of the screen) and a graphics portion. Text is 8 rows by 32 columns; the graphics portion is 128 pixels by 256 pixels.

**Split-Screen Mode: Text-Low**—The same as Split-Screen: Text-High except that the text is at the bottom one-third of the screen, with graphics at the top.

**High-Resolution Mode**—A full-screen graphics mode with 192 X 256 pixels.

**Multicolor Mode**—Rather than in characters or pixels, the screen displays in blocks, with 48 X 64 blocks.

• **Screen Margins**—CALL MARGINS enables you to redefine the screen margins and thus define a text "window" on the screen.

• **Editing**—TI Extended BASIC II includes an EDIT Mode with a wide range of editing capabilities, including the ability to edit line numbers.

• **Functions**—In addition to the fundamental numeric and string functions, TI Extended BASIC II includes:

MAX, which returns the larger of two numbers.

MIN, which returns the smaller of two numbers.

PI, which returns the value of  $\pi$ .

FREESPACE(0), which returns the amount of unused memory space.

VALHEX, which returns the decimal (base 10) value of a hexadecimal (base 16) number.

TERMCHAR, which returns the key code of the key or key combination (such as ENTER, PROC'D, BACK, BEGIN) that terminated the last INPUT, LINPUT, or ACCEPT statement.

• **Arrays**—Arrays may have up to seven dimensions. The OPTION BASE instruction sets the lowest subscript to either zero or one.

• **String Handling**—The RPT\$ function allows the repetition of a string.

• **Error Handling**—With ON WARNING, ON ERROR, and ON BREAK, you can choose what action is taken if there is a minor error, a major error, or a breakpoint, respectively. RETURN can be used with error handling. The CALL ERR statement can be used to ascertain the nature of an error that occurs in a program.

lat degree headlines

Beginning of general copy  
(bold weight)

• **RUN as a Statement**—RUN can be used as a statement as well as a command. RUN also enables you to specify which program to run. As a result, one program can load and run another program from a diskette. Thus, you can write programs of almost unlimited size by breaking them into pieces and letting each segment run the next.

• **Power/Up Program Execution**—When TI Extended BASIC II is first chosen, it searches for a program named LOAD on the diskette in disk drive 1. If that program exists, it is automatically loaded into memory and run.

• **Multiple Statement Lines**—TI Extended BASIC II allows for more than one statement on a line. This feature speeds program execution, saves memory, and allows logical units (for example, FOR-NEXT loops) to be on a single line.

• **Program Protection**—The PROTECTED option with the SAVE command protects your programs by coding (encrypting) them so they can only be loaded and run. Because a PROTECTED program cannot be edited, listed, or saved, this option prevents changes in your programs. However, PROTECTED files can be copied from one diskette to another using the Disk Manager Solid State Cartridge.

• **IF THEN ELSE**—The IF THEN ELSE statement allows statements as the consequences of the comparison. This permits statements such as

```
IF X>4 THEN GOSUB 240 ELSE  
X=X+1
```

• **Multiple Assignments**—TI Extended BASIC II enables you to assign a value to more than one variable in a LET statement, saving statements and permitting more efficient programming. For example: R,V,X = 1

• **Trailing Remarks**—In addition to the REM statement, trailing remarks can be added to the ends of lines in TI Extended BASIC II, allowing detailed internal documentation of programs. An exclamation point (!) begins each trailing remark. For example,

```
COUNT=1 ! LOOP COUNTER IS SET  
TO 1
```

• **Assembly Language Support**—TMS9905 (of which TMS9900 code is a subset) assembly language subprograms may be loaded and run. The subprograms INIT, LOAD, LINK, PEEK, PEEKV, and POKEV are used to access assembly language subprograms.

• **Information**—The FREESPACE(0) function returns the amount of memory that remains unused in your computer. The VERSION subprogram returns a value that indicates the version of BASIC that is in use. The CHARPAT subprogram returns a character string indicating the pattern that defines a character.

• **DISPLAY VARIABLE 80 Format**—Files in DISPLAY VARIABLE 80 format, created by the LIST command or a text editing or word processing program, may be loaded with the OLD command.

• **Key-Chirp**—TI Extended BASIC II enables you to turn on an audible "chirp" that sounds whenever a key is pressed. Enter the following command to turn on the key-chirp:

```
CALL LOAD(VALHEX("84BD"),1)
```

Enter the following command to turn the key-chirp off:

```
CALL LOAD(VALHEX("84BD"),0)
```

File No

10

11

# Differences Between TI Extended BASIC II and Previous Versions of TI BASIC

Listed below are the notable differences between TI Extended BASIC II and the two prior versions of this programming language, TI BASIC and TI Extended BASIC. Due to the enhancements added to TI Extended BASIC II, programs written in either of the two earlier versions of the language may require slight

modifications to run properly on the Computer 99/8. If you are experienced in using these versions, read this section carefully in order to avoid unexpected results with TI Extended BASIC II.

## Speed of Execution

Because TI Extended BASIC II executes more rapidly, delay loops do not delay a program for as long a duration. Similarly, if a negative duration is used with the SOUND subprogram, the sounds are executed approximately three times as quickly.

If you wish to run programs on the 99/8 at 99/4A Home Computer speed, refer to the examples under LOAD in the

Reference Guide.

## The Value Stack

The Value Stack, which contains information such as GOSUB entries, FOR-NEXT entries, and temporary values, is limited to 3840 bytes. This means that an infinite GOSUB loop can no longer ascertain the amount of available program space. This operation is now performed by the built-in FREESPACE(0) function.

## HCHAR and VCHAR

The HCHAR and VCHAR subprograms cause characters to be printed outside the margins, but do not cause them to be scrolled (moved up the screen) as subsequent data items are entered. These characters may be erased with the CLEAR subprogram.

Displaying a character using PRINT in the lower-right corner of the current window will cause a scroll.

## The PROTECTED Option

If a program saved with the PROTECT option stops executing, either normally, at a breakpoint, or because of an error, it is erased from memory. To execute the program again, it must be reloaded into memory from an external device. Note that when a PROTECTED program is loaded into memory, it begins execution immediately.

## Numeric Data Types (to replace IN)

TI Extended BASIC II can store two types of numeric data: integer and real. To be stored as an integer, a number must be whole and within the range - 32768 to 32767. Numbers outside this range must be stored as real data.

An integer data value requires three bytes of storage when written to a cassette or diskette file in INTERNAL format. A real data value requires 7 bytes of storage when written to a file in INTERNAL format. Because TI BASIC and TI Extended BASIC do not support INTEGER data types, INTEGER values printed to a file in INTERNAL format by TI Extended BASIC II cannot be input as numeric values by the two prior versions of BASIC. To create files that are compatible with all versions of TI

BASIC, convert any integer data values to real values before printing them to a file.

Note: Integer data values can be input as string values by TI BASIC or TI Extended BASIC. Because these values are stored in INTERNAL format, they are not directly interpretable as numeric values.

## String Length

BASIC II permits strings of up to 4090 characters in length. TI BASIC and TI Extended BASIC permit strings of up to 255 characters in length.

## Record Length

TI Extended BASIC II allows diskette data files created on the Disk Drive/Controller to have VARIABLE records with record-lengths up to 4090 bytes. TI BASIC and TI Extended BASIC allow VARIABLE records to be only 254 bytes long. Diskette data files with FIXED length records are limited to 255 bytes (the same as TI BASIC and TI Extended BASIC).

TI Extended BASIC II allows cassette data files to have FIXED length records up to 4032 bytes. (Cassette data files cannot have VARIABLE length records.) TI BASIC and TI Extended BASIC limit cassette data files to records of 192 bytes or less.

TI Extended BASIC II files with record lengths longer than those permitted by TI BASIC or TI Extended BASIC cannot be input by those versions of BASIC.

## Assembly-Language Programs

Assembly-language programs that use absolute addressing with TI BASIC variables or with program loading do not execute in TI Extended BASIC II. Relocatable assembly-language programs will, however, execute correctly, although NUMREF may return an integer value instead of a floating-point value. This is denoted by >6B in FAC + 2. In addition, a string reference error may occur with STRREF because STRREF only allows strings of up to 255 characters. TI Extended BASIC II allows strings of up to 4090 characters.

Also, some assembly-language programs containing references to invalid addresses (such as 80xx and 82xx) are executable in TI BASIC and TI Extended BASIC. These programs do not work in TI Extended BASIC II.

All POKES and PEEKs as used in previous BASICs fail in TI Extended BASIC II. In addition, the ability to PEEK and/or LOAD values into program memory is not predictable; thus, LOAD cannot be used to modify a program in memory.

1st degree headlines

Beginning of general copy  
(bold weight)

### Memory Allocation

The INIT subprogram with no parameter allocates 8K bytes of memory; if a parameter is specified, up to 24,336 bytes may be allocated.

TI Extended BASIC allocates approximately 6K bytes of memory.

The Editor/Assembler with TI BASIC allocates 32K bytes of memory.

### Software Cartridges

CALL to routines contained in a plug-in cartridge are not accessible from TI Extended BASIC II. Thus, programs that use the Personal Record Keeping cartridge will not execute properly. However, all of the features which were included in the Personal Record Keeping cartridge are now available in TI Extended BASIC II.

Text to Speech cannot be accessed from TI Extended BASIC II with the Terminal Emulator II cartridge. The following OPEN commands for the Terminal Emulator II do not work from TI Extended BASIC II:

OPEN #1: "SPEECH", OUTPUT

OPEN #2: "ALPHON", INTERNAL

Otherwise, the Terminal Emulator II cartridge functions normally.

14

### Reserved Words

The list of reserved words has been expanded in TI Extended BASIC II. The following words are additions to the TI BASIC reserved word list. Words preceded by an asterisk (\*) are additions to the TI Extended BASIC reserved word list. Reserved words may not be used as variable names.

ACCEPT	NOT
ALL	NUMERIC
*ALPHA	OR
AND	PI
AT	*REAL
BEEP	RPT\$
DIGIT	SIZE
ERASE	SUBEND
ERROR	SUBEXT
*FREESPACE	*TERMCHAR
IMAGE	UALPHA
*INTEGER	USING
*LALPHA	*VALIEX
LINPUT	VALIDATE
MAX	WARNING
MERGE	XOR
MIN	

### Modes in TI Extended BASIC II

There are two main operating modes in TI Extended BASIC II: Command Mode and Run Mode.

There are also two other modes used when entering and editing program lines: Number Mode and Edit Mode.

Command Mode is the mode entered when you choose TI Extended BASIC II on the master selection list. In the Command Mode you may enter program lines or any instruction that can be used as a command.

In Run Mode, a TI Extended BASIC II program is executed. This is done with the RUN command. You can stop a running program only by pressing CLEAR (FCTN 4), which causes a breakpoint, or with QUIT (FCTN =). (Note: QUIT also erases the entire program, returns you to the master title screen, and may delete information from some of your files.) It is recommended that you press CLEAR to stop a program from running and that you use the BYE command to leave TI Extended BASIC II. QUIT should be used primarily to stop a Solid State Cartridge program.

Number Mode (entered with the NUMBER command) generates line numbers for you as you are entering a program. Number Mode can be used also to edit existing program lines. Refer to the NUMBER command for details about this mode.

Edit Mode is used to edit existing program lines. It is discussed in detail under the next heading, "Error Correction and Editing."

PUT IN RULES

15



## Error Correction and Editing

As discussed briefly in the *Getting Started* book, there are many different ways to correct errors (generally, typographical errors rather than program errors). The type of error and when it occurs dictate what method is used to correct it.

and edit

To correct errors discovered before you press ENTER:

- Backspace with **←** (FCTN 5) and type the correct character(s) over the incorrect one(s) or use the **SPACE BAR** to blank out incorrect characters.
- Press **ERASE** (FCTN 3) to erase the entire line and start over again.
- Press **CLEAR** (FCTN 4) to indicate that the line is not to be accepted by the computer. The line scrolls up on the screen.
- Use **DELETE** (FCTN 1) or **INSERT** (FCTN 2); these functions are explained under "Edit Mode."

To correct errors discovered after you press ENTER:

- Type the line over again, using the same line number. When you enter the corrected line, it will replace the incorrect line. If you type just the line number and then press ENTER, the line is deleted from your program.
- Press **REDO** (FCTN 8) to edit the line most recently input. **REDO** can be used if you discover an error after you press ENTER but before another line is input. **REDO** puts the computer in Edit Mode.

Type the line number and press either **↑** (FCTN E) or **↓** (FCTN X) to enter Edit Mode. The line specified is then displayed on the screen and can be edited.

### Edit Mode

Edit Mode is used to edit existing lines of a TI Extended BASIC II program. You can change a line by typing a new line, by typing over part of the old line, or by using the editing keys discussed below.

To enter the Edit Mode, type the number of the line you want to edit and press either **↑** (FCTN E) or **↓** (FCTN X). The computer displays the specified program line with the cursor positioned after the space following the line number. The prompt symbol (**>**) is not displayed.

If you change the line number of a program statement, you create a new program statement with the line number you specify. The original program statement remains unchanged.

**Special Editing Keys** **←** (FCTN S) and **→** (FCTN D) are cursor-control keys. Use these keys to move the cursor to the point in the line that you want to edit.

**DELETE** (FCTN 1) deletes the character in the cursor position. All characters to the right of the cursor move to the left.

**INSERT** (FCTN 2) inserts characters that you type at the cursor position. The character at the cursor position and all characters to the right of the cursor move to the right as you type. Insertion continues with each character typed until ENTER or one of the other special editing keys is pressed. You may lose characters if they move so far to the right that they no longer fit in the program line (256 characters).

**ERASE** (FCTN 3) erases the program line currently displayed (including the line number) from the screen and from the input buffer. You can then enter a new line (with the same line number or a different one). **ERASE** does not delete the program line from memory. To delete the line, you must type the line number and then press ENTER.

**CLEAR** (FCTN 4) causes the computer to leave the Edit Mode. The program line you were editing is not changed.

**REDO** (FCTN 8) causes the program line most recently input to be displayed; thus you can make corrections without

having to retype the entire line. This can be especially helpful if you make an error while editing a program line, causing the computer not to accept it.

**ENTER** causes the computer to accept the changes you have made to the line you are editing and to leave Edit Mode.

**↑** (FCTN E) causes the computer to accept the changes you have made to the line you are editing, display the next lower-numbered line in your program, and remain in Edit Mode. If no lower-numbered line exists in your program, the computer leaves Edit Mode.

**↓** (FCTN X) causes the computer to accept the changes you have made to the line you are editing, display the next higher-numbered line in your program, and remain in Edit Mode. If no higher-numbered line exists in your program, the computer leaves Edit Mode.

When you press ENTER, **↑**, or **↓** while in Edit Mode, the computer:

- Accepts the valid program line, regardless of the cursor position.
- Deletes a program line if all but the line number has been erased.
- Accepts no changes in a program line if any change causes an error, and leaves Edit Mode.

## Overview of TI Extended BASIC II

This section briefly describes the TI Extended BASIC II commands, statements, functions, and subprograms and suggests ways to use them.

- Commands
- Assignments and Input

### Beginning of specific copy

#### Commands

Commands instruct the computer to perform a task immediately (that is, as soon as you press ENTER), whereas statements (with line numbers) are executed when a program is run. In TI Extended BASIC II, many instructions (such as RUN, BREAK, UNBREAK, TRACE, UNTRACE, and DELETE) can be used as either commands or statements.

#### NEW

To clear a program from memory and prepare the computer to accept a new program, use the NEW command. Programs are also removed from memory by the OLD command and the RUN command when used with a filename.

#### NUMBER and RESEQUENCE

When you are entering a program, the computer assigns line numbers for you if you enter the NUMBER command. If you wish to resequence the line numbers of a program after it is written, use the RESEQUENCE command.

#### LIST

To review a program that you have entered, use the LIST command. The program, or a portion of the program, can be listed on the screen or to a peripheral device.

- Output
- Functions, Subroutines, and Subprograms
- Sound, Speech, and Color
- Sprites
- Debugging
- Error Handling
- Pre-scan—!@P— and !@P+

#### RUN

The RUN command instructs the computer to perform or "execute" a program. The RUN command may be followed by either a line number (to start program execution at a specific line) or by a string expression (to load and execute a program from a peripheral device).

#### TRACE, UNTRACE, BREAK, UNBREAK, and CONTINUE

All of these commands are related to "debugging" a program, which is the process of finding errors in a program. These commands are discussed further in the "Debugging" section.

#### SAVE, OLD, MERGE, and DELETE

When you have finished working with a program, you will generally want to store it on a mass-storage medium, such as a diskette, cassette, etc. The SAVE command, followed by the name of the storage device and a program name, performs this task for you. Then, when you wish to reuse, list, edit, or change a program, you can load it into memory with the OLD command. If a program is saved using the MERGE option, you can combine it with a program already in memory with the MERGE command. When you have no further use for a program that has been saved on diskette, you can remove it with the DELETE command.

#### FREESPACE(0)

The FREESPACE(0) function enables you to ascertain the amount of available memory. This helps you decide whether to continue to add program lines or to end the current program and start a second program. The first program can run the second program by using RUN as a statement.

#### BYE

When you have finished using TI Extended BASIC II, use the BYE command to return to the master title screen.

#### Assignments and Input

This section discusses statements in TI Extended BASIC II that assign values to variables and enter data into programs.

#### LET and READ

If you know what values are to be assigned to variables, use LET or READ statements. LET is used when you are assigning a fairly small number of values or are calculating values to be assigned. READ is used, in conjunction with DATA and RESTORE, when you are assigning a large number of values.

#### INPUT and INPUT

When you want the user of the program to enter values, it is customary to display a prompt that asks for the necessary information. INPUT enables you to display a prompt and accept input. INPUT allows only the entry of values at the bottom of the screen and

does not check to see that the data entered is the type of information the program expects. The final limitation on INPUT is that commas and quotation marks affect what is entered. With INPUT, there is no editing of what is input, so commas and quotation marks can be entered. INPUT and INPUT are also used to input data from files on a mass-storage medium.

#### ACCEPT

ACCEPT allows input from most screen positions. Using ACCEPT eliminates the necessity of entering data at the bottom of the screen and the "scrolling" of the INPUT statement. However, ACCEPT doesn't allow a prompt as the INPUT statement does. Therefore, ACCEPT is generally preceded by a PRINT or DISPLAY statement to tell the user the type of entry that is required. ACCEPT is for screen and keyboard use only.

#### CALL KEY and CALL JOYST

If pressing a single key is all that the program user is required to do, then CALL KEY can be used. For example, if a Y for "yes" or N for "no" is the required response, use the CALL KEY statement to accept the entry. CALL KEY does not display a character on the screen. It scans the keyboard or a portion of the keyboard to see if a key has been pressed.

1st degree headlines

Beginning of general copy  
(bold weight)

The major limitation of CALL KEY is that only a single keystroke is accepted. The data is not recorded as a string character, but as an ASCII code. (See Appendices A and D for a list of the codes used.) If you wish to show the character that was pressed, you must use the CALL HCHAR or CALL VCHAR subprograms, or the CHR\$ function in conjunction with the DISPLAY or PRINT commands.

The input from a Joystick Controller can be used with CALL JOYST. As with CALL KEY, the data is not displayed, and no scrolling takes place.

**CALL CHARPAT, CALL COINC, CALL DISTANCE, CALL ERR, CALL GCHAR, CALL POSITION, CALL SAY, CALL SPGET, and CALL VERSION.**

Each of these statements returns information. CALL CHARPAT assigns a value that specifies the pattern of a character. CALL COINC assigns a value to tell if all sprites, two sprites, or one sprite and a particular pixel are at or near the same location on the screen. CALL DISTANCE indicates the distance between two sprites or a sprite and a pixel on the screen. CALL ERR returns the last error that occurred and the line where it occurred. CALL GCHAR reads what character is at a given screen location, returns the status of a screen pixel, or returns the color of a block. CALL POSITION reads where a sprite is on the

screen. CALL SAY enables you to instruct the computer to produce speech. CALL SPGET assigns the coded value of a speech word or phrase to a variable to be used with CALL SAY. CALL VERSION indicates the version of BASIC in use.

#### **FOR TO STEP and NEXT.**

The FOR TO STEP statement sets the value of a variable so that it can be used to control the number of times a loop is executed. Each time NEXT is encountered, the value of that variable is changed (incremented or decremented). After the last loop has been completed, the variable has a value that is the first value outside the range specified in the FOR TO STEP statement.

#### **Output**

This section discusses the TI Extended BASIC II statements used to output data during program execution. Usually, output consists of displaying information on the screen, printing data on a printer, or saving data on an external device. However, output can also involve changing the color of the screen, changing the colors of characters, drawing graphics, making sounds, speaking, or sending data to peripheral devices.

**PRINT, DISPLAY, PRINT USING, DISPLAY USING, and IMAGE.**

(The two most frequently used output

statements are PRINT and DISPLAY. The print separators (comma, semicolon, and colon) and the TAB function are used to control the placement of information as it is output. PRINT displays items at the bottom of the screen and scrolls them upward, one line at a time. With DISPLAY, you can display data almost anywhere on the screen without scrolling. DISPLAY can also clear the screen, erase characters on a line, and sound a "beep."

PRINT USING and DISPLAY USING are like PRINT and DISPLAY except that the format of the printed or displayed characters is determined by the USING clause, possibly in conjunction with an IMAGE statement. The USING clause allows exact control of the display format. PRINT and PRINT USING, possibly in conjunction with IMAGE, are the only output statements that can be used to send data to an external device.

#### **CALL HCHAR, CALL VCHAR, and CALL SPRITE.**

CALL HCHAR and CALL VCHAR place a character at any screen position and optionally repeat it horizontally or vertically. CALL SPRITE displays "sprites" on the screen. Sprites are graphics that can be moved smoothly in any direction and changed in pattern, size, and color. CALL SPRITE and the other statements related to sprites are discussed later in this section.

#### **CALL SCREEN and CALL COLOR.**

In addition to displaying characters and data on the screen, you can change the color of the screen and the colors of the characters. CALL SCREEN sets the screen color. CALL COLOR specifies the foreground and background colors of characters or the color of sprites.

#### **CALL DRAW, CALL DRAWTO, CALL FILL, and CALL DCOLOR.**

These are graphics subprograms which enable you to plot graphics and add color to them on the screen. CALL DRAW and CALL DRAWTO draw or erase lines between specified pixels, thus making elaborate figures or drawings possible. (The screen is comprised of a grid of 256x192 individual points. Each of these points is called a "pixel.") CALL FILL colors the area surrounding a specified pixel. CALL DCOLOR sets the graphics colors that are used by CALL DRAW, CALL DRAWTO, CALL FILL, CALL HCHAR, and CALL VCHAR.

#### **CALL SOUND, CALL SAY, and CALL SPGET.**

CALL SOUND causes sounds to be generated. A wide range of tones and noise is available. In addition, CALL SAY (possibly used with CALL SPGET) makes the computer speak.

Other subprograms ("CALL" instructions) have other special uses. See "Built-in Subprograms" for a complete list of the built-in subprograms.

Folio

20

21

1st degree headlines

Beginning of general copy  
(bold weight)

### Functions, Subroutines, and Subprograms

TI Extended BASIC II provides extensive functions and subprograms for handling numbers and characters. In addition, you may construct your own functions and write your own subprograms and subroutines.

Functions are TI Extended BASIC II language elements that return a value, usually based on parameters given to

the function. Functions are used within TI Extended BASIC II statements. Many functions are mathematical in nature; others control or affect the result produced by the statements in which they occur.

#### Built-In Functions

The following table briefly explains each built-in function. The "Data Type" column gives the data type(s) of each function.

Function	Data Type	Value Returned and Comments
ABS	Real or Integer	Absolute value of a numeric expression.
ASC	Integer	The ASCII code of the first character of a string expression.
ATN	R <sub>N</sub>	Trigonometric arctangent (given in radians) of a numeric expression.
CHR\$	String	Character that corresponds to an ASCII code.
COS	R <sub>A</sub>	Trigonometric cosine of an angle whose measurement is given in radians.
EOF	I <sub>A</sub>	End of file condition of a file.
EXP	R <sub>N</sub>	Exponential value (e <sup>x</sup> ) of a numeric expression.
FREESPACE	R <sub>A</sub>	Number of bytes of free (unused) memory space.
INT	I <sub>N</sub>	Integer value of a numeric expression.
LEN	I <sub>N</sub>	Number of characters in a string expression.

Endo

22

LIGHT BLUE ON TOP

Function	Data Type	Value Returned and Comments
LOG	R <sub>N</sub>	Natural logarithm of a numeric expression.
MAX	R or I <sub>N</sub>	Larger of two numeric expressions.
MIN	R or I <sub>N</sub>	Smaller of two numeric expressions.
PI	R <sub>N</sub>	$\pi$ with a value of 3.141592653589.
POS	I <sub>N</sub>	Position of the first occurrence of one string expression within another.
REC	I <sub>N</sub>	Current record position in a file.
RND	R <sub>N</sub>	Random number from 0 to 1.
RPT\$	S <sub>N</sub>	String expression equal to a number of copies of a string expression concatenated together.
SEG\$	S <sub>N</sub>	Substring of a string expression, starting at a specified point in that string and ending after a certain number of characters.
SGN	I <sub>N</sub>	Sign of a numeric expression.
SIN	R <sub>N</sub>	Trigonometric sine of an angle whose measurement is given in radians.
SQR	R <sub>N</sub>	Square root of a numeric expression.
STR\$	S <sub>N</sub>	String equivalent of a numeric expression.
TAB	-	Position for the next item in the print-list of a PRINT or DISPLAY instruction.
TAN	R <sub>N</sub>	Trigonometric tangent of an angle whose measurement is given in radians.
TERMCHAR	I <sub>N</sub>	The key code of the key pressed to enter input in an INPUT, LINPUT, or ACCEPT statement.
VAL	R <sub>N</sub>	Numeric value of a string expression that represents a number.
VALHEX	I <sub>N</sub>	The decimal (base 10) value of a hexadecimal (base 16) number.

23

1st degree headlines

Beginning of general copy  
(bold weight)

### User-defined Functions

You can also define your own functions with the DEF instruction. Functions up to one input line in length may be defined with up to seven parameters. Parameters are the values used to evaluate the function and are enclosed in parentheses.

Longer functions may be constructed by having new functions refer to previously defined functions. However, long functions might be more efficiently handled with subroutines or subprograms.

### Subroutines

A subroutine is a series of statements designed to perform a task and is normally used in a program when it performs a task several times. GOSUB and ON GOSUB are used to "call" subroutines; RETURN is used to return to the main body of the program. Using GOSUB or ON GOSUB enables you to write a program segment once and simply call it when you wish to execute it. You do not have to retype the program segment every time you want to use it. The subroutine can use the values of any variable in the program and change those values.

### Built-in Subprograms

Built-in subprograms are TI Extended BASIC II elements that perform special operations. They are accessed with the CALL statement.

Built-in subprograms perform many different tasks. Some of the subprograms affect the display and ascertain what key has been pressed on the keyboard.

#### Built-in Subprogram Action and Comments

CLEAR Clears the screen.

COLOR Specifies the color of characters, the color of blocks, or the color of sprites.

GCHAR Returns the ASCII code of the character at a screen position, the status of a screen pixel (High-Resolution Mode), or the color of a block (Multicolor Mode).

HCHAR Displays a character on the screen and optionally repeats it horizontally.

JOYST Returns values indicating the position of the Joystick Controllers (optional).

KEY Returns a code indicating the key that has been pressed.

#### Built-in Subprogram Action and Comments

SCREEN Specifies the color of the screen.

VCHAR Displays a character on the screen and optionally repeats it vertically.

MARGINS Redefines the current screen margins, thus defining a text "window" on the screen.

Built-in subprograms can also define and control sprites.

#### Built-in Subprogram Action and Comments

CHAR Specifies the pattern for a sprite or a graphic.

COINC Determines whether all sprites, two sprites, or a sprite and a pixel are at or near the same location on the screen.

COLOR Specifies the color of a sprite, characters, or blocks.

DELSprite Deletes a sprite.

#### Built-in Subprogram Action and Comments

DISTANCE Returns the distance between two sprites or a sprite and a location.

LOCATE Specifies the position of a sprite.

MAGNIFY Changes the size of a sprite.

MOTION Specifies the motion of a sprite.

PATTERN Specifies the character that defines a sprite.

POSITION Returns the position of a sprite.

SPRITE Defines a sprite, specifying its defining character, color, position, and motion.

Folio

24

25

1st degree headlines

Beginning of general copy  
(bold weight)

Graphics subprograms enable you to use the special graphics capabilities of TI Extended BASIC II.

Built-in Subprogram	Action and Comments
---------------------	---------------------

GRAPHICS	Specifies one of the six graphics modes.
----------	--

DCOLOR	Specifies the colors that are used by the DRAW, DRAWTO, FILL, HCHAR, and VCHAR subprograms.
--------	---

DRAW	Draws or erases lines between specified points (pixels).
------	--

DRAWTO	Draws or erases lines between the current position and a specified pixel.
--------	---

FILL	Colors the area surrounding a specified pixel.
------	--

Sound and speech subprograms cause the computer to generate sounds or speech.

Built-in Subprogram	Action and Comments
---------------------	---------------------

SAY	Causes the computer to speak words.
-----	-------------------------------------

SOUND	Generates tones and noises.
-------	-----------------------------

SPGET	Retrieves the codes that make speech.
-------	---------------------------------------

There are built-in subprograms used only with assembly-language subprograms written in TMS9900 (or TMS9900) assembly language.

Built-in Subprogram	Action and Comments
---------------------	---------------------

INIT	Allocates memory space for assembly-language subprograms.
------	---

LINK	Establishes a link between BASIC and the assembly-language subprogram and optionally passes parameters.
------	---

LOAD	Loads assembly-language object code and/or pokes byte values into memory.
------	---

PEEK	Reads byte values from memory.
------	--------------------------------

PEEKV	Reads byte values from video memory.
-------	--------------------------------------

POKEV	Pokes byte values into video memory.
-------	--------------------------------------

Finally, there are some miscellaneous built-in subprograms:

Built-in Subprogram	Action and Comments
---------------------	---------------------

CHARPAT	Returns a value that identifies the pattern of a character.
---------	---

CHARSET	Resets all characters to their original predefined patterns and colors.
---------	---

ERR	Returns values that give information about an error that has occurred.
-----	--

VERSION	Returns a value that identifies the version of TI Extended BASIC II that is being used.
---------	---

Folio

27

1st degree headlines

Beginning of general copy  
(bold weight)

#### User-written Subprograms

You may write your own subprograms. They are a series of statements designed to perform a task. They may be used, like a subroutine, in a program when you expect to perform the task several times.

Subprograms are especially useful when you want to perform the same

task in several different programs. Using the MERGE option when you save a subprogram enables you to include it in other programs.

When a subprogram is included in a program, it must follow the main program. The structure of a program must be as follows:

The subprograms you write are not part of the main program. Unlike subroutines, they cannot use the values of variables in the main program. Thus, any values that are needed must be supplied by the parameter list in the CALL statement. Variable names within the subprogram may duplicate those in the main program or other subprograms without affecting the values of the variables in the main program or other subprograms.

**Note:** If you use an END or STOP statement inside the subprogram, program execution is stopped and values are not reassigned to the parameters. No error message or warning is given.

Subprograms cannot be recursive; that is, they can call other subprograms, but must not call themselves, either directly or indirectly.

SUBEND must be the last statement in a subprogram. When that statement is executed, control returns to the statement immediately following the CALL statement that called the subprogram. Control may also be returned by the SUBEXIT statement.

Start of Main Program

Subprogram Calls

End of Main Program

The program will stop here without a STOP or END statement.

Start of First Subprogram

End of First Subprogram

Nothing may appear between subprograms except remarks and an END statement.

Start of Second Subprogram

End of Second Subprogram

Only remarks and END may appear after the subprograms.

End of Program

Subprograms are executed with the CALL instruction, followed by the subprogram's name and an optional list of parameters and values. The first line of a subprogram is SUB, followed by the

name of the subprogram, and optionally followed by a list of parameters. The last line of each subprogram must include the SUBEND command.

File

28

29

Let's go headlines

Beginning of general copy  
(bold weight)

### Sound, Speech, and Color

You may highlight important sections of your program's output through the use of sounds, speech, and colors. These features make the program easier and more interesting to use.

**CALL SOUND** SOUND sends out tones or noise; these can be programmed for music or sound effects. Tones may be output in lengths of from .001 to 4.25 seconds at volumes from 0 (loudest) to 30 (softest). The frequency range is from 119 (A below low C) to well above the range of human hearing. In addition, eight noises are available. Up to three tones and one noise may be produced at the same time. Appendix H lists the frequencies that are used to produce the musical notes.

**CALL SAY** and **CALL SPGET** SAY produces speech. You can choose from among 373 letters, numbers, words, phrases, and compounds (listed in Appendix L). For example, **SOME + THING** produces "something" and **THERE + FOUR** produces "therefore." A pound sign is used on either end of a predefined phrase (more than one word); for example,

**CALL SAY("TEXAS INSTRUMENTS")**

**SPGET** is used to retrieve the speech codes that produce speech. These patterns can then be used to produce more

natural speech and can be used to change words. Because making new words is a lengthy and complex process, it is not discussed in this book. However, suffixes can be added rather simply. Appendix M tells how to add the suffixes **ING**, **S**, and **ED** to any word, so that words such as **ANSWERING**, **ANSWERS**, **ANSWERED**, **INSTRUCTING**, **INSTRUCTS**, and **INSTRUCTED** are included in the computer's vocabulary.

**CALL COLOR** and **CALL SCREEN** COLOR changes the colors of character sets, individual characters, or blocks (depending on the graphics mode) and determines sprite colors. **SCREEN** specifies the color of the screen as one of the sixteen colors available on the TI Computer 99/8.

### Sprites

Sprites are graphics that can be displayed and moved on the screen. One advantage that sprites have over other characters is that they can be at any of 40,152 positions of 102 pixel rows and 256 pixel columns rather than one of the 768 positions of 24 rows and 32 columns used by statements such as **CALL VCHAR** and **CALL HCHAR**. Because of this greater resolution, sprites can move more smoothly than characters. Also, once set in motion, sprites can continue to move without further program control.

**CALL SPRITE** The **SPRITE** subprogram defines sprites. This subprogram specifies the character pattern that sprites use, their color, their position, and, optionally, their motion.

**CALL CHAR** and **CALL MAGNIFY** Although you can use any of the predefined characters, numbers 32-223, as a sprite, the **CHAR** subprogram is generally used to define a new pattern for a sprite. The **MAGNIFY** subprogram controls the resolution and size of sprites.

**CALL COLOR**, **CALL LOCATE**, **CALL PATTERN**, and **CALL MOTION** Once a sprite is set up, it can be altered by various subprograms. **COLOR** changes the color of a sprite. **LOCATE** moves the sprite to a new position.

**PATTERN** changes the character that defines a sprite. **MOTION** alters the motion of a sprite.

**CALL COINC**, **CALL DISTANCE**, and **CALL POSITION** Three subprograms provide information about sprites while a program is running. **COINC** returns a value that indicates whether two sprites or a sprite and a point on the screen are at or near the same place. **DISTANCE** returns a value that specifies the distance between two sprites or a sprite and a point on the screen. **POSITION** returns values that indicate the position of a sprite.

**CALL DELSPRITE** and **CALL LOCATE** **CALL DELSPRITE** enables you to delete sprites. If you prefer, you may "hide" sprites by locating them off the bottom of the screen (pixel rows 193-256) with **CALL LOCATE**.

**CALL GRAPHICS** **CALL GRAPHICS** specifies one of the six graphics modes: Pattern Mode, Text Mode, Split-Screen: Text-High, Split-Screen: Text-Low, High-Resolution Mode, and Multicolor Mode. **DCOLOR** specifies the colors that are used by the **DRAW**, **DRAWTO**, **FILL**, **HCHAR**, and **VCHAR** subprograms. **DRAW** draws or erases lines between specified points (pixels). **DRAWTO** draws or erases lines between the current position and a specified point. **FILL** colors the area surrounding a specified pixel.

Follow

30

31



1st degree headlines

Beginning of general copy  
(bold weight)

MOVE ERROR  
MESSAGE  
UP TO NEXT  
COLUMN

## Debugging

Debugging a program is the process of finding and correcting logical or typographical errors in a program. BREAK, CONTINUE, TRACE, ON BREAK, UNBREAK, and UNTRACE are most often used in debugging.

**BREAK, ON BREAK, CONTINUE, and UNBREAK** BREAK causes the computer to stop program execution so that you can check the values of variables or change their values. BREAK also resets characters to their standard colors (black on transparent), restores the standard screen color (cyan), restores all characters (0-255) to their standard representation, and deletes sprites.

**ON BREAK** tells the computer what to do if a break occurs. You can use this statement to tell the computer to ignore breakpoints that you have established in the program. CONTINUE causes the computer to continue program execution after a breakpoint. UNBREAK cancels breakpoints set with BREAK. Note: If you have used ON BREAK NEXT, the computer will not stop when you press CLEAR. You will have to turn the computer off or press QUIT to interrupt such a program.

## Error Handling

You may include statements in a program to handle errors that occur while

the program is running.

**CALL ERR, ON ERROR, ON WARNING, and RETURN** CALL ERR returns information indicating where an error has occurred and what the error is. Appendix W lists the error codes returned. ON ERROR specifies what the computer does if a condition arises that would normally cause an error message to be issued and the program to stop. ON WARNING specifies what the computer does if a condition arises that would normally cause a warning message to be issued. RETURN can be used with ON ERROR in addition to its use with GOSUB. RETURN repeats execution of the statement that caused the error, returns to the statement following the one that caused the error, or transfers control to some other part of the program to avoid the error that has occurred.

**TRACE and UNTRACE** TRACE causes the computer to display each line number before the statement(s) on that line is (are) executed. Using this statement enables you to follow the sequence of execution of a program. UNTRACE cancels the operation of TRACE.

## Pre-scan—!@P— and !@P+

After you enter RUN to start a program, you may notice a pause before the program actually begins. This pause is the time the computer takes to "pre-scan" your program to establish memory space for variables, arrays, and data. Then the computer proceeds through each instruction, performs the appropriate functions, and establishes variable values. Since the time required to pre-scan depends on the length of the program, you may want to decrease the pre-scan pause, particularly if you have a long program.

With the pre-scan statements, !@P— and !@P+, you can specify which statements are to be pre-scanned. These statements have the following uses:

!@P— Turns pre-scan "off."  
!@P+ Turns pre-scan "on."

Note that a lower-case "p" may be used in place of the upper-case "P."

Because the purpose of the pre-scan is to set memory space for variables, only those instructions that contain the first reference to the variables must be pre-scanned. Therefore, many other instructions in your program do not require a pre-scan. By carefully using the !@P— and !@P+ statements, you reduce the computer's pre-scan to only

the instructions that require pre-scanning, thus reducing the length of the pause before program execution.

Careful program planning is required to minimize the statements that must be pre-scanned. Generally, the following statements should be included in the pre-scan.

- The first DATA statement.
- All DIM, INTEGER, and REAL statements.
- The first use of each variable and/or array (including the OPTION BASE statement, if used).
- All DEF statements for user-defined functions.
- All SUB statements and SUBEND statements.

Note that a variable in a user-defined (SUB) subprogram is considered to be unique from any other variable used elsewhere in your program, even though the name and value may be the same. Therefore, each variable used in a user-defined subprogram must be included in the pre-scan.

let agree headlines

Beginning of general copy  
(hold weight)

To use the pre-scan option, first be certain that your completed program runs successfully. Then, at the beginning of a group of statements, use a !@P - statement to "turn off" the pre-scan. The following statements will not be pre-scanned, allowing the execution of your program to begin more quickly. Any statements related to variable names (not previously referenced during pre-scan) return a syntax error if the pre-scan is "off." Note that !@P - cannot be followed by another statement in a multiple-statement line, though it can be used at the end of a multiple-statement line.

To resume the pre-scan, simply insert a !@P + statement. This causes the pre-scan to "turn on" and memory space for variables may be set.

You may choose to use the pre-scan feature several times throughout your program. By selectively turning the pre-scan on and off, your program will begin to execute more quickly. The effectiveness of controlling the pre-scan is more noticeable in large programs than in small programs.

The following examples illustrate the use of pre-scan statements in an existing program:

Original program:

```
>100 CALL CLEAR
>110 CALL CHAR(96, "FFFFFFFF
FFFFFFFF")
>120 CALL CHAR(42, "0F0F0F0F
FOF0F0F")
>130
>140
>150
>160 CALL HCHAR(12, 17, 42)
>170 CALL VCHAR(14, 17, 96)
>180 DELAY=0
>190 FOR DELAY=1 TO 500
>200 NEXT DELAY
>210 DATA 3
>220
>230
>240
```

With pre-scan control added:

```
>10 DATA 3
>100 CALL CLEAR
>110 CALL CHAR(96, "FFFFFFFF
FFFFFFFF")
>120 CALL CHAR(42, "0F0F0F0F
FOF0F0F")
>125 !@P-
>130
>140
>150
>155 !@P+
>160 CALL HCHAR(12, 17, 42),
```

```
>170 CALL VCHAR(14, 17, 96)
>180 DELAY=0
>185 !@P
>190 FOR DELAY=1 TO 500
>200 NEXT DELAY
>210
>220
>230
```

Notice that the first DATA statement (as line 210) has been moved to the beginning of the program (line 10) so that it is included in the pre-scan. By including statements 125, 155, and 185, the pre-scan is turned off and on and off again. This causes the program to begin executing more quickly.

You have the ability to "trick" the computer into establishing memory space for CALL statements, as well as variable-related statements, without actually performing those statements. To do this, simply use a GOTO instruction in your program. The following example demonstrates the original program adapted with a pre-scan and a GOTO statement.

With GOTO added:

```
>10 DATA 3
>20 GOTO 100::DELAY::CALL CH
AR::CALL CLEAR::CALL HCHAR:
[[CALL VCHAR::!@P-
>100 CALL CLEAR
>110 CALL CHAR(96, "FFFFFFFFFF
FFFFFF")
```

```
>120 CALL CHAR(42, "0F0F0F0F
FOF0F0F")
>140
>150
>155
>160 CALL HCHAR(12, 17, 42)
>170 CALL VCHAR(14, 17, 96)
>190 FOR DELAY=1 TO 500
>200 NEXT DELAY
>210
>220
>230
```

The GOTO method causes the necessary memory space to be reserved in line 20. Note that the statements in line 20 are never executed; however, the subprograms referenced there are executed later on in the program. Thus, as shown in the preceding and following examples, you can put all of your variable references together and your subprogram calls need not be syntactically correct. This is the most efficient use of the pre-scan option.

```
>100 GOTO 170::X,Y,ALPHA,BET
A,Z=DELTA::DIM B(10,10)
>110 CALL KEY::CALL HCHAR::C
ALL,CFAR::CALL MYSUR
>120 DATA 1,3,STRING
>130 DEF F(X)=1-X*SIN(X)
>160 !@P-
>170 REM BEGIN EXECUTION-HERE
>180
>190
>200
```

Full

34

35

## TI Extended BASIC II Conventions

This section discusses the format required of TI Extended BASIC II programs and the ways in which TI Extended BASIC II functions.

### Running a Program on Powerup

If a program named LOAD is on the diskette in disk drive 1 when TI Extended BASIC II is selected from the main selection list, that program is automatically loaded and run. The effect is the same as if you had entered `PRINT "DISK1 LOAD"`. If the program does not exist, there is a momentary delay while TI Extended BASIC II looks for it.

### Program Size

The maximum program size for the Computer 00/8 without memory expansion is 63,320 bytes. This includes only program text, and not variable, string, or array space.

### Files

Files are groups of data put on external devices. The most common files are on data-storage devices, but data sent through external devices such as the RS-232 interface are also considered to be files by TI Extended BASIC II.

### Line Numbers

Line numbers are required in TI Extended BASIC II programs. Line numbers specify the order in which lines are executed and are used to identify what lines to execute next when using `IF THEN ELSE`, `GOTO`, `GOSUB`, `ON`

`ERROR`, `ON GOTO`, and `ON GOSUB`. You also can use line numbers with `BREAK`, `LIST`, `NUM`, `RESTORE`, `RETURN`, and `RUN`. A line number may be any integer from 1 through 32767.

The computer automatically generates line numbers if you use the `NUM` (NUMBER) command. When not followed by a line number, `NUM` provides line numbers starting at 100, incrementing each subsequent line by 10. You can resequence line numbers with the `RES` (RESEQUENCE) command.

### Lines

Lines may be up to 255 characters long, including the line number and spaces. If you have reached the end of a line, additional characters you enter replace the 255th character. A line is "crunched" (reduced to an internal format) when `ENTER` is pressed by replacing each keyword (`PRINT`, `STEP`, etc.) with a single character item, called a "token." A "crunched" line must be no longer than 160 characters, or a Line too long error will occur.

### Upper-case and Lower-case

TI Extended BASIC II commands, statements, functions, and variable names may be entered as lower-case characters. The computer automatically converts them to upper case. However, if a command, statement, or

function contains a string constant (such as the file specification in `OPEN`), lower-case characters in that string constant are not converted to upper case. Likewise, lower-case characters that are parts of `DATA`, `IMAGE`, and `REM` statements are not converted to upper case.

### Special Symbols

Special symbols separate statements and remarks on the same line. A line of TI Extended BASIC II consists of a line number, one or more statements, and an optional remark. For example:

```
>100 FOR A=1 TO 100:PRINT A
;SQR(A)::NEXT A !PRINT SQUA
RE ROOTS
```

The statement separator symbol, a double colon (`::`), is used to separate statements on the same line. The trailing remark symbol, an exclamation point (`!`), is used to separate an explanatory remark from the rest of the line. Remarks are not executed when the program is run; they are a means of internally documenting the program.

### Spaces

Spaces are required in TI Extended BASIC II between the elements that make up statements to enable the computer to distinguish variable names from TI Extended BASIC II elements.

However, spaces are not required before or after relational expressions or before or after the trailing remark symbol or the statement separator symbol. You can insert extra spaces when entering commands and statements, but they are deleted by TI Extended BASIC II when these lines are stored in memory. When listing programs, TI Extended BASIC II may add spaces around the trailing remark symbol and statement separator symbol.

1st degree headlines

Beginning of general copy  
(bold weight)

### Numeric Data

TI Extended BASIC II can store two types of numeric data: integer and real. To be stored as an integer, a number must be whole and within the range of -32768 to 32767. Numbers outside this range must be stored as real data.

An integer data item requires only two bytes of memory for storage; a real data item requires eight bytes of memory for storage. Because integers use less storage space, they can be manipulated faster than real numbers by the computer. This quality makes it advantageous to use integers whenever possible. For example, a FOR-NEXT loop that uses an integer control variable will execute more quickly than the same FOR-NEXT loop using a real control variable. Note that it is possible to use integer data only when a value is a whole number within the range given above.

The computer always stores numeric data in real format unless specifically instructed to do otherwise. The computer stores a value as an integer when either of the following cases is true:

- The value is assigned to a variable that has been defined previously as an integer variable (using the INTEGER statement).

- The value is produced by a function that has a predefined integer data

type. For example, EOF and VALHEX return integer values.

If a numeric value outside the range -32768 to 32767 is assigned to an integer variable, the warning message **INTEGER OVERFLOW** in line-number is displayed. If a numeric value is within the range -32768 to 32767, but is not a whole number, it is rounded to the nearest integer prior to being assigned to the variable, as illustrated in the following table. (The examples assume that the variable X has been defined as an integer variable.)

Assignment Operation	Value Assigned to X
LET X= 5.4	5
LET X= 5.6	6
LET X=-1.2	-1
LET X=-2.7	-3

When the result of an arithmetic operation is assigned to an integer variable, the computer applies the following rules:

- If the operands are numeric constants or real variables, the computer rounds the result to the nearest integer prior to assigning the result to the integer variable.

Example:

```
>100 INTEGER A
>110 A=5/3
>120 DISPLAY A
```

The value assigned to A is 2. Because the operands in the calculation are constants, the result (1.666666667) is rounded prior to being assigned to A.

- If the operands are integer variables, the computer truncates any digits to the right of the decimal point prior to assigning the result to the integer variable.

Example:

```
>100 INTEGER A,B,X
>110 A=5
>120 B=3
>130 X=A/B
>140 DISPLAY X
```

Because the operands in the calculation are integers, integer division (which discards the remainder) occurs; therefore, the value assigned to X is 1. (PRINT A/B would also cause the computer to print 1).

When the result of an arithmetic operation is assigned to a real variable, it is stored as a real number, even though the result may be a whole number within the range -32768 to 32767.

### Numeric Constants

Unless assigned to integer variables or used as arguments in integer functions, numeric constants are always stored as real (not integer) data and can be entered with any number of digits. However, they are rounded to 13 or 14 digits by the computer due to the internal storage method used by the computer and are normally displayed as a maximum of 10 digits. For extremely large or small numbers, it is usually more convenient to use scientific notation to enter numbers. The computer normally uses scientific notation when printing very large or small numbers.

In scientific notation, a number is given as a mantissa (a number with one place to the left of the decimal point) multiplied by 10 raised to an integer power. 15 is expressed in scientific notation as  $1.5 \times 10^1$ . 150 is expressed as  $1.5 \times 10^2$ ; -1,500 is expressed as  $-1.5 \times 10^3$ ; 150,789,000,000,000 is expressed as  $1.50789 \times 10^{14}$ ; and 0.150789 is expressed as  $1.50789 \times 10^{-1}$ . In TI Extended BASIC II, the " $\times 10^n$ " is represented by "E." Thus,  $1.5 \times 10^{15}$  becomes  $1.5E+15$ .

Folio

38

39



1st degree headlines

Beginning of general copy  
(bold weight)

>100 A=B=7

Sets A equal to -1 if B is equal to 7, and to 0 if B is not equal to 7. There is no effect on B. Note that this is not the same as the usual arithmetic meaning of  $A = B = 7$ , nor is it equivalent to the assignment sequence  $A, B = 7$ .

#### Logical Expressions

Logical expressions are used with relational expressions. The logical operators are AND, OR, NOT, and XOR. If true, logical expressions are given a nonzero value. If false, they are given a value of 0. The order of precedence for logical expressions, from highest to lowest priority, is NOT, XOR, AND, and OR. Logical expressions have a lower precedence than relational expressions.

• A logical expression using AND is true if both its left and right clauses are true.

• A logical expression using OR is true if one or both of its clauses are true.

• A logical expression using NOT is true if the clause following it is not true.

• A logical expression using XOR (exclusive or) is true if one but not both of its clauses is true.

The following examples illustrate the use of logical expressions:

>100 IF 3<4 AND 5<6 THEN L=7

Sets L equal to 7 since 3 is less than 4 and 5 is less than 6.

>100 IF 3<4 AND 5>6 THEN L=7

Does not set L equal to 7 because 3 is less than 4, but 5 is not greater than 6.

>100 IF 3<4 OR 5>6 THEN L=7

Sets L equal to 7 because 3 is less than 4.

>100 IF 3<4 XOR 5>6 THEN L=7

Sets L equal to 7 because 3 is less than 4 and 5 is not greater than 6.

>100 IF 3<4 XOR 5<6 THEN L=7

Does not set L equal to 7 because 3 is less than 4 and 5 is less than 6.

>100 IF NOT 3=4 THEN L=7

Sets L equal to 7 because 3 is not equal to 4.

>100 IF NOT 3=4 AND (NOT 6=5 XOR 2=2) THEN 200

Does not pass control to line 200, because although it is true that 3 is not equal to 4, it is true both that 6 is not equal to 5 and that 2 is equal to 2, so the clause in parentheses is not true.

>100 IF (A OR B) AND (C XOR D) THEN 200

Passes control to line 200 if either A or B or both A and B are true (equal to -1), and C or D, but not both C and D are true (equal to -1).

The logical operators can also be used directly on numbers. They convert the numbers to binary notation, perform the designated operation on a bit level, and then convert the result back to decimal representation. A more detailed discussion of the use of logical operators with numbers can be found in a mathematics or engineering text dealing with logic.

The numbers must be from -32,768 to 32,767, represented in binary notation as from 1000000000000000 to 0111111111111111. Negative numbers are given in 2's complement form, which means that zeroes are reversed to ones and vice versa, with a 1 added to the most significant bit (the first digit in the binary number). In binary notation, each place is an additional power of 2 rather than an additional power of 10 as in decimal notation. The following shows numbers in both decimal and binary notation.

Decimal Place	Binary Place
- 100 10 1 - 1024 512 256 128 64 32 16 8 4 2 1	
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	
0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0	
0 2 5 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1	
- 0 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1	

The above is the equivalent to

$1_{10} = 0000000000000001_2 = 1_2$   
 $6_{10} = 0000000000000110_2 = 110_2$   
 $-25_{10} = 00000000000011001_2 = 11001_2$   
 $-13_{10} = 1111111111110011_2$

Philo

42

43

1st degree headlines

Beginning of general copy  
(bold weight)

AND places a 1 in the corresponding binary position if there is a 1 in both the number preceding and following it. Otherwise it places a 0.

OR places a 1 in the corresponding binary position if there is a 1 in either the number preceding it or following it or both. Otherwise it places a 0.

XOR places a 1 in the corresponding binary position if there is a 1 in either

the number preceding it or following it but not both. Otherwise it places a 0.

NOT places a 1 in the corresponding binary position if there is a zero in the number following it. Otherwise it places a zero.

The following illustrate the result of the logical operators when used on numbers.

Decimal	Binary	Decimal	Binary
A: 1	0000000000000001	A: 1	0000000000000001
B: 2	0000000000000010	B: 3	0000000000000011
A AND B: 0	0000000000000000	A AND B: 1	0000000000000001
A: 6	0000000000000110	A: 47	000000000101111
B: 5	0000000000000101	B: 62	000000000111110
A AND B: 4	0000000000000100	A AND B: 46	000000000101110

Decimal	Binary	Decimal	Binary
A: 1	0000000000000001	A: 1	0000000000000001
B: 2	0000000000000010	B: 3	0000000000000011
A OR B: 3	0000000000000011	A OR B: 3	0000000000000011
A: 6	0000000000000110	A: 47	000000000101111
B: 5	0000000000000101	B: 62	000000000111110
A OR B: 7	0000000000000111	A OR B: 63	000000000111111

Decimal	Binary	Decimal	Binary
A: 1	0000000000000001	A: 1	0000000000000001
B: 2	0000000000000010	B: 3	0000000000000011
A XOR B: 3	0000000000000011	A XOR B: 2	0000000000000010
A: 6	0000000000000110	A: 47	000000000101111
B: 5	0000000000000101	B: 62	000000000111110
A XOR B: 3	0000000000000011	A XOR B: 17	000000000010001

Decimal	Binary	Decimal	Binary
A: 1	0000000000000001	A: 2	0000000000000010
NOT A: -2	1111111111111110	NOT A: -3	1111111111111101
A: 6	0000000000000110	A: 47	000000000101111
NOT A: -7	1111111111111001	NOT A: -48	1111111111010000

File

45

1st degree headlines

Beginning of general copy  
(bold weight)

**String Expressions** String expressions are constructed from string variables, string constants, and function references using the operation for concatenation (&) to combine strings. If a constructed string exceeds a length of 4000 characters, the extra characters on the right are truncated and a warning message is displayed. The following is an example of concatenation:

>A\$="HI "&" THERE!"

"HI" and "THERE" are concatenated; assigned to A\$.

#### Margins and the Screen Window

TI Extended BASIC II enables you to specify the screen margins that define the area of the screen that can be accessed by certain instructions. The area within the margins is called the screen window, or just the window.

You can define the setting of four margins: left, right, top, and bottom. Each margin setting indicates the number of character positions the margin is indented from the edge of the screen.

To change the current margins, use the MARGINS subprogram. See the MARGINS subprogram for a detailed explanation of margins and the screen window.

When you enter Extended BASIC II, the left and right margins are set to 2, and the top and bottom margins are set to 0. When you use the GRAPHICS subprogram to change the graphics mode, these margins (2,2,0,0) are restored.

You can define a window in Pattern and Text Modes, and in the text portion of the screen in the Split-Screen Modes. You cannot define a window in High-Resolution or Multicolor Mode, or in the graphics portion of the screen in the Split-Screen Modes.

The ACCEPT, DISPLAY, DISPLAY USING, INPUT, LINPUT, PRINT, and PRINT USING instructions treat the top row of the window as row 1 and the leftmost column of the window as column 1, regardless of the screen position of the window. These instructions consider all row and column specifications to be relative to the upper-left corner of the window (the intersection of row 1 and column 1), rather than to the upper-left corner of the screen.

The INPUT, LINPUT, PRINT, PRINT USING, DISPLAY, and DISPLAY USING instructions, which cause scrolling, affect the screen window only.

The CLEAR subprogram, which displays spaces, affects both the screen window and the graphics portion of the screen.

The VCHAR, GCHAR, and HCHAR subprograms and the graphics portion of the screen, are not affected by margins.



1st degree headlines

Beginning of general copy  
(bold weight)

#### Graphics Modes

Six graphics modes are available in TI Extended BASIC II: Pattern Mode, Text Mode, two Split-Screen Modes (Text-High and Text-Low), High-Resolution Mode, and Multicolor Mode. You can select the graphics mode that offers you the combination of text and graphics capabilities that best suits the particular needs of your program.

The Split-Screen Modes divide the screen into text and graphics portions. The text portion fills one-third of the screen; the graphics portion fills the remaining two-thirds of the screen. In Text-High Mode, the text portion is the upper third of the screen; in Text-Low Mode, the text portion is the lower third of the screen.

When you enter Extended BASIC II, the computer is in Pattern Mode. To change the current graphics mode, use the GRAPHICS subprogram. See the GRAPHICS subprogram for detailed descriptions of the graphics modes.

The following table summarizes the capabilities of the graphics modes, and indicates which graphics instructions can be used in each mode. The characteristics of the text and graphics portions of the screen in the Split-Screen Modes are listed independently.

	Pattern	Text	Split: Text	Split: Graphics	High- Res.	Multi- color
Rows	24 char.	24 char.	8 char.	128 pixels	192 pixels	48 blocks
Columns	32 char.	40 char.	32 char.	256 pixels	256 pixels	64 blocks
Sprites	yes	no effect	yes	yes	yes	yes
DOOLOR	no effect	no effect	no effect	yes	yes	no effect
DRAW, DRAWTO, FILL	error returned	error returned	no effect	yes	yes	error returned
GCHAR	yes	yes	no effect	yes	yes	yes
HCHAR, VCHAR	yes	yes	no effect	yes	yes	no effect
ACCEPT, DISPLAY, DISPLAY USING, INPUT, LINPUT, PRINT, PRINT USING	yes	yes	yes	invisible*	invisible*	invisible*

\*Text "displayed" is not visible. If, for example you use LINPUT, the computer will pause to accept input, even though you are unable to see a prompt.

File

517

48

49

1st degree headlines

Beginning of general copy  
(bold weight)

#### Defaults

When you enter TI Extended BASIC II, certain features are set with predefined, or default, values. These features retain their default values unless you execute an instruction that changes them.

The default values in Extended BASIC II are as follows:

- Graphics Mode—The computer is in Pattern Mode.
- Margins—The left and right margins are set to 2; the top and bottom margins are set to 0.
- Screen—The screen window and the screen margins are cleared (filled with spaces).
- Display Characters—All alphabetic characters are displayed on the screen as upper-case characters.
- Character Definitions—ASCII codes 32-127 are defined as the standard character set (see Appendix E). The cursor is defined as number 30. Codes 128-223 are defined as the same standard character set as codes 32-127. The remaining codes (0-29, 31, 224-255) are defined as spaces.
- Colors—The foreground color is set to black; the background color is set to

transparent. The graphics colors, used by the DRAW, DRAWTO, and FILL subprograms, are also set to black on transparent. The screen or border color is set to cyan.

- Sprites—No sprites are defined. The sprite magnification level is set to 1 (single-size, unmagnified).
- Key Unit—In the KEY subprogram, the key unit is set to 6 (the BASIC emulator keyboard).
- Current Position—The position used as a starting point by the DRAWTO subprogram is set to 1,1 (pixel row 1, pixel column 1).
- Sound—All four sound generators (the three music generators and the noise generator) are turned off.
- Alphabetic Characters—All alphabetic characters are set to uppercase.

All the default values (except the keyunit in the KEY subprogram) are restored when you use the GRAPHICS subprogram to change the graphics mode. (The graphics mode, of course, is changed to the specified value.)

Some values are affected when your program stops running, whether by reaching the end of the program (an END or STOP statement or the highest-numbered program statement), encountering a breakpoint, pressing CLEAR (FCTN 4), or due to a program error:

- Character definitions, foreground and background colors, screen (border) color, sprites, sprite magnification level, and the key unit in CALL KEY are restored to their default values.
- The graphics colors and current position remain unchanged.
- Sound remains unchanged unless a breakpoint is encountered.
- If the computer is in High-Resolution or Multicolor Mode, the default graphics mode and screen margins are restored. If the computer is in Pattern, Text, or one of the Split-Screen Modes, the graphics mode and margins remain unchanged.

Follow

50

51

## File Commands

The Computer 99/8 has the capability to store both programs and data on peripheral (accessory) devices. You can later load and use these files with your computer as often as you wish, and delete them when you no longer need them.

With the file-processing capabilities of your computer, you can save important information, create procedures to update data, and avoid retyping your programs into the computer each time you wish to use them. TI Extended BASIC II provides an extensive range of file-processing features, including sequential and relative (random) file processing, fixed and variable-length records, and display and internal data formats.

The following is a brief overview of the file-processing commands and statements of TI Extended BASIC II. For more complete information, refer to the descriptions of each in the reference section of this manual.

OLD	Loads a program into memory.
SAVE	Saves a program in memory to a specified device.
RUN	Begins program execution. If a file specification is included, the program is loaded from the specified device, then executed.
OPEN	Establishes a link between the computer and the specified device.
PRINT	Outputs data to a previously opened file.

INPUT	Inputs data from a previously opened file.
LINPUT	A more specialized form of INPUT that inputs an entire record at a time from a previously opened file.
EOF	Tests for an end-of-file condition.
REC	Returns the number of the next record to be accessed by the specified file.
RESTORE	Repositions the record counter.
CLOSE	Removes the previously established link between the computer and the specified device.
DELETE	Deletes a file from the specified device.
LIST	Lists the program on the screen. If a file-specification is included, the computer sends a listing of the program to a specified device.
MERGE	Combines a program from the specified device with the program currently in memory.

Some of the commands listed above may only operate with peripherals that support certain file attributes. The default record length is supplied by the peripheral.

Refer to the manual that comes with the specific peripheral for complete details on the record length and commands supported by that device.

### Cassette Interface File Commands

The OLD, SAVE, RUN, OPEN, PRINT, INPUT, LINPUT, RESTORE, and CLOSE commands are supported by the cassette interface.

The DELETE command has no effect on files stored on cassette. The message is displayed, but no action is taken on the file.

PRESS CASSETTE STOP  
THEN PRESS ENTER

The EOF function cannot be used with cassette files. It is recommended that you place a "dummy" value as the last record and then check for this value (using an IF THEN statement) as you input data from the file. A common dummy value is several nines (999999). This value, when input, indicates that there is no more data in the file.

The LIST and MERGE commands cannot be used with cassette files, because

the cassette interface will not accept VARIABLE length records, which are required by these commands.

**Note:** When you use the OLD, SAVE, RUN, OPEN, RESTORE, and CLOSE commands and statements with the "CSI" file specification, the computer will display instructions on the screen for you to follow.

### Using the OPEN Statement with the Cassette Interface

The following is a list of file attributes specific to files opened on cassette.

- file-specification (required)—CSI
- file-organization (optional)—SEQUENTIAL
- open-mode (required)—INPUT or OUTPUT
- record-type (required)—FIXED
- record-length (optional)—Cassette files have record-lengths that are less than or equal to 4032 bytes. If you specify a record-length that is not a multiple of 64, its value is increased to the next greater multiple of 64. If you do not specify a record-length, it is assumed to be 64.
- The maximum length record that can be read by the INPUT statement is 128 bytes. The LINPUT statement can read records of up to 4032 bytes. **Note:** LIN-

## Beginning of general copy (bold weight)

### File Specification copy (regular weight)

The *file-specification* used with some file-processing commands and statements is a string expression that specifies the device and, sometimes, filename, special options, or commands used by that device. If *file-specification* is a constant, some commands, such as DELETE, LIST, OPEN, and RUN, require that it be enclosed in quotation marks. Quotation marks are optional with the MERGE, OLD, and SAVE

There are three Input/Output ports available on the 99/8 computer:

- The 50-pin peripheral port on the side of the console is used to attach future peripherals.
- The cassette port on the back of the console can be used to attach a standard audio cassette recorder.
- The HEX-BUS™ interface enables you to attach a wide variety of special low-cost peripherals to the Computer 99/8.

### General Format

The general format for the *file-specification* is:

*device-name* [*software-options*] [*filename*]

where *device-name* is a string expression (all alphabetic characters must be upper case), *software-options* are special commands that are recognized by the device being accessed (not all peripherals have *software-options*), *filename* is the name you assign to a file so that it can be stored on a mass-storage device.

There are some general conventions that apply for the cassette and HEX-BUS™ interfaces. The *file-specification* for the 50-pin peripheral port are explained in the manuals that come with the peripherals designed for that interface.

### Cassette File Specification

The *file-specification* for the cassette interface is CSI. (The cassette interface does not support any *software-options* or use *filenames*.)

### HEX-BUS™ File Specification

The *file-specification* for HEX-BUS peripherals can have one of the following formats:

HEXBUS[*subcommand*].*device-number*

or

HEXBUS.*subcommand*

where:

The *subcommand* is a special instruction that must be used in conjunction with a valid command to the HEX-BUS™ interface. The valid *subcommands* are listed below. Each is explained later in this section (except the Transfer Raw Data *subcommand*, which is explained in Appendix R, and the Slave mode *subcommand*, explained in Appendix S).

VE	Verify
FORMAT MEDIA	Format storage medium
RD	Reset Bus
CA	Catalog
TR	Transfer raw data
SL	Slave mode

The *device-number* is an integer specifying the HEX-BUS peripheral you wish to access. Refer to the manual that comes with each peripheral for more information on *device-number*.

The *software-options* are special commands that are recognized by the device being accessed. Not all peripherals have *software-options*.

The *filename* is the name you assign to a file so that it can be stored on a mass-storage device. Refer to the manual that comes with each peripheral for any restrictions that may apply to the *filename* when used with that peripheral.

The first format

HEXBUS[*subcommand*].*device-number* [*software-options*]

is normally used with HEX-BUS peripherals not designed for mass storage, such as the RS232 interface, the Printer/Plotter, and the Modem.

The second format

HEXBUS[*subcommand*].*device-number* [*filename*]

is normally used with the HEX-BUS™ peripherals designed for mass storage, such as the Disk Drive/Controller 5102.

1st degree headlines

Beginning of general copy  
(fold weight)

### The third format

#### HEXBUS.subcommand

is used to access the Transfer Raw Data capability of the HEX-BUS interface. The capability to transfer raw data is described in Appendix R. This capability is generally used only in very advanced applications.

Some HEX-BUS™ peripherals can be accessed by using the general format for file specification, described earlier. However, the subcommands cannot be used when this method of access is chosen.

HEX-BUS peripherals that may be accessed with this alternate method of addressing are the Disk Drive/Controller, the RS232, and the HEX-BUS Modem. The following chart lists the device names and their corresponding device numbers.

Device-name	Device-number
DSK.disk-name	100.disk-name
DSK1	101
DSK2	102
DSK3	103
DSK4	104
RS232	20
RS232/1	20
RS232/2	70

Note that the device-name for the Modem is RS232/2.

The Printer/Plotter, with a device-number of 10, and the parallel port, with a device-number of 50, cannot be accessed with this alternate method of addressing.

### Examples

The following examples illustrate that the device-name method is interchangeable with the HEXBUS.device-number method.

```
>100 OPEN #1:"DSK.DATA.MYFILE"
>100 OPEN #1:"HEXBUS.100.DATA.MYFILE"
```

Opens a file with a filename of MYFILE on the disk named DATA.

```
>SAVE DSK1.PROG1
>SAVE HEXBUS.101.PROG1
```

Saves a file with the filename PROG1 on disk drive 1.

```
>LIST "RS232/1.BA=1200"
>LIST "HEXBUS.20.BA=1200"
```

Lists a program to the RS232, and uses a software-option to set the baud rate at 1200.

**HEX-BUS™ Subcommands** The VE subcommand verifies that the information in memory is the same as the information stored on a mass-storage device. If there is a discrepancy, an I/O error message is returned.

When used to verify data, the VE subcommand has the following format:

```
OPEN #file-number:"HEXBUS.VE.device-number.filename"
```

The file should be opened in the OUTPUT mode. The PRINT statement functions as a verify command when the file is opened in the VE mode. No data can be written to the file while it is opened in the VE mode. Also, a file cannot be opened in the normal mode and the VE mode at the same time.

When used to verify a program, the VE subcommand has the following format:

```
SAVE HEXBUS.VE.device-number.filename
```

The VE subcommand cannot be used to verify a program saved with the PROTECTED option.

Full

56

57

lat degree headlines

Beginning of general copy  
(bold weight)

Examples:

```
>SAVE HEXBUS.101.PROGRAM1
>SAVE HEXBUS.VE.101.PROGRAM1
```

Saves a file on *device-number* and then verifies it.

The following program creates a data file and then verifies it.

```
>100 OPEN #1:"HEXBUS.101.EXA
MPLE",INTERNAL,FIXED 20
>110 FOR X=1 TO 10
>120 PRINT #1:X,X*2
>130 NEXT X
>140 CLOSE #1
>150 OPEN #10:"HEXBUS.VE.101
.EXAMPLE",INTERNAL,FIXED 20
>160 FOR X=1 TO 10
>170 PRINT #10:X,X*2
>180 NEXT X
>190 CLOSE #10
>200 DISPLAY "NO ERRORS"
>210 END
```

Lines 100-140 open a file called EXAM-  
PLE on a disk, print 10 records, and  
close the file.

Lines 150-190 open the same file in the  
VE mode, verify that the data was writ-  
ten correctly to the file, and close the  
file.

Line 200 displays a message that there  
were no errors found. If an error is  
detected, an I/O error occurs. For more

information on I/O errors, refer to Ap-  
pendix U.

The **FORMAT MEDIA** subcommand  
directs a **HEX-BUS™** mass-storage  
peripheral to prepare the storage  
medium to accept information. This  
process is commonly called  
"initialization."

The **FORMAT MEDIA** subcommand has  
the following format:

OPEN #*file-number*:"HEXBUS.FOR-  
MAT MEDIA,*device-number*"

CLOSE #*file-number*

The **FORMAT MEDIA** subcommand  
must be used before any information  
can be written onto the storage  
medium. If the subcommand is used  
later, any information already stored on  
the storage medium is erased. **FORMAT**  
**MEDIA** is most often used in the Com-  
mand mode, rather than as a statement  
in a program.

After the storage medium is initialized,  
execute a **CLOSE** statement. If you do  
not close the *file-number* assigned by  
the **FORMAT MEDIA** subcommand,  
that *file-number* cannot be used in  
another **OPEN** statement.

Note: Executing the **FORMAT MEDIA**  
subcommand on a diskette with an  
open file causes an error condition.

Example:

```
>OPEN #1:"HEXBUS.FORMAT MEDI
A.101"
>CLOSE #1
```

The **RB (RESET BUS)** subcommand  
closes all files opened to **HEX-BUS™**  
peripherals, and "resets" the com-  
munications lines. Using **RB** is a good  
practice if a program that accesses a  
mass-storage device ends with an error.

The **RB** subcommand has the following  
format:

OPEN #*file-number*:"HEXBUS.RB.0"

CLOSE #*file-number*

When **RB** is used, the indicator lights on  
each peripheral should blink momen-  
tarily. **RB** is the only subcommand  
which addresses all **HEX-BUS**  
peripherals at once.

The **CA (CATALOG)** subcommand is  
reserved for use with future mass-  
storage devices. **CA** enables you to ac-  
cess information stored on the directory  
of a storage medium.

The **CA** subcommand has the following  
format:

OPEN #*file-number*:"HEX-  
BUS.CA,*device-number*"

CLOSE #*file-number*

File

58

59

1st degree headlines

Beginning of general copy  
(bold weight)

The following program, written in TI Extended BASIC II, demonstrates how to interpret directory information from a device. The device-number (a number from 1-7 inclusive) is entered from the keyboard.

```
>100 ON ERROR 420
>110 DISPLAY AT(10,1)ERASE A
LL:"WHICH DRIVE?"
>120 ACCEPT AT(10,14)BEEP VA
LDATE("1234567")SIZE(1):DR$
>130 DEV$="HEXBUS.CA."&DR$:
EMPTY$="Y"
>140 DISPLAY AT(23,1)ERASE A
LL:"FILENAME MAX L REC #
FLG"
>150 OPEN #1:DEV$,INPUT,FI
XE
D 18
>160 INPUT #1:A$
>170 EMPTY$="N"
>180 DISPLAY TAB(1);SEG$(A$,
2,12);
>190 A1=ASC(SEG$(A$,14,1))
>200 A2=ASC(SEG$(A$,15,1))
>210 DISPLAY TAB(13);A2*256+
A1;
>220 A1=ASC(SEG$(A$,16,1))
>230 A2=ASC(SEG$(A$,17,1))
>240 DISPLAY TAB(19);A2*256+
A1;
>250 ACTIVE$,LAST$="N"
>260 DATATYPE$="D"
>270 IF A1<128 THEN 300
>280 A1=A1-128
>290 ACTIVE$=A$
>300 IF A1<64 THEN 330
>310 A1=A1-64
```

```
>320 LAST$="Y"
>330 IF A1<16 THEN 350
>340 DATATYPE$="I"
>350 DISPLAY TAB(26);ACTIVE$
,DATATYPE$
>360 IF LAST$="N" THEN 160
>370 CLOSE #1
>380 DISPLAY "CATALOG ANOTHE
R DRIVE? (Y/N)"
>390 ACCEPT VALIDATE("YyNn")
>SIZE(1):C$
>400 IF C$="Y" OR C$="y" THEN
N 100
>410 END
>420 CALL FRR(Q,R,S,IN)
>430 IF LN=160 THEN IF EMPTY
$="Y" THEN PRINT TAB(5);"MED
IUM EMPTY":RETURN 370 ELSE
RETURN 370 ELSE RETURN
>440 STOP
```

In the display, FILENAME is the name you assigned when you saved the data or program. MAX L is the length of the longest record in that file. REC # is the number of records in that file. FLG is a two-character flag that indicates the active status of the file and the storage format of the information. The first character is either A for ACTIVE files or N for deleted files. The second character is either D for DISPLAY format or I for INTERNAL format.

## Reference Section for TI Extended BASIC II

This section contains detailed explanations of all TI Extended BASIC II instructions, in alphabetical order.

Instruction explanations include:

- **Heading**—The name of the instruction, in large type. If the instruction is a built-in subprogram, the word "subprogram" immediately follows the instruction name. If the instruction is a built-in function, the word "function" immediately follows the instruction name. If the instruction name is an abbreviation or acronym, it is followed by a brief expansion.

If the instruction name consists of more than one word, the words may or may not appear consecutively in actual use. See the format line to ascertain whether the words are used consecutively or are separated by one or more items.

- **Format**—The form and syntax for valid instruction use. The format line defines the elements that can appear as part of an instruction, and indicates acceptable syntax and instruction options. See the next page for a description of the notational conventions used in format lines.

- **Type**—For built-in functions, this lists the possible data type(s), such as numeric (real or integer) and string. Returned values are of the same data type as the particular function.

- **Purpose**—A brief description of the function of the instruction.

- **Cross Reference**—A list of related instructions. You may find it helpful to refer to the specified instruction descriptions for a more complete understanding of the use of an instruction. Cross references are not indicated for all instructions.

- **Description**—A detailed discussion of the instruction. The description includes a full explanation of each item in the format line, as well as exceptions, options, possible errors, and other relevant information.

- **Examples**—Illustrations of the use of the instruction. In the examples, the prompt character (>) indicates lines that you enter. Lines that the computer displays on the screen are not preceded by a prompt character.

1st degree headlines

Beginning of general copy  
(bold weight)

#### Notational Conventions for Format Lines

The explanation for each instruction begins with a format line that specifies the proper form and syntax to be used. Format lines follow these notational conventions.

- Words in CAPITAL letters are the "keywords" used for each instruction.
- *Italicized words* are the general descriptions of the types of items that you must supply. The description section includes an explanation of each italicized item in the format line.
- Braces ( { } ) indicate a choice between alternate forms. You must use only one of the enclosed items.
- Brackets ( [ ] ) indicate that the enclosed item is optional.
- Ellipsis points ( ... ) indicate that the preceding item(s) can be repeated.
- All punctuation except for brackets and braces (such as periods, commas, quotation marks, colons, or parentheses) must be included where shown.

#### Examples

**Format**  
CALL HCHAR(*row*,*column*,*character-code*  
[,*number-of-repetitions*])

This format line shows that the keywords are immediately followed by an open parenthesis. Then you must supply numbers for *row*, *column*, and *character-code*, separated by commas. The *number-of-repetitions* is enclosed in brackets; thus, it is optional. If you do not include *number-of-repetitions*, enter a close parenthesis after *character-code*; if you do include *number-of-repetitions*, it must be preceded by a comma and followed by a close parenthesis.

**Format**  
(DIM  
[*data-type*] *array name*(*integer-1*  
[,...*integer-7*])(*array-name*...)

Two different keywords can be used to reserve memory space for an array: either the DIM keyword or a *data-type* keyword (INTEGER or REAL).

The keyword is followed by one space, the array name, an open parenthesis, and at least one *integer* to specify the size of the array. Up to seven *integers* may optionally be used (indicated by the general description *integer-7*); if you

include more than one *integer*, they must be separated by commas. A close parenthesis is required after the last *integer*.

If you want to reserve space for more arrays in the same statement, a comma is required, followed by the next array name, and so forth.



# Beginning of general copy (bold weight)

## General Descriptions

The *italicized* general descriptions used in format lines vary according to the requirements of the particular instruction. Common general descriptions include:

- *numeric-expression*—A numeric constant, variable, or expression.
- *string-expression*—A string constant, variable, or expression. Some instructions require you to enclose a string constant in quotation marks, as indicated in the instruction description.
- *numeric- or string-variable*—A "return" variable. Rather than specifying a value that you want the computer to use, a return variable is used by the computer to store the result of an operation so that you may access that result.
- *parameter*—A value that is "passed" from one instruction to another. Parameters are used in the CALL instruction to pass values to a built-in or user-defined subprogram, so that you can use the same subprogram to operate on different values. Similarly, parameters are used in the DEF instruction so that you can pass values to user-written functions.
- *list*—One or more of the specified items. If the *list* includes more than

one item, use commas to separate the items

- *line-number*—The number preceding a TI Extended BASIC II program statement.
- *file-specification*—A string expression specifying a particular file or device. The exact nature of the *file-specification* depends on the peripheral device. For more information see "File Specifications" and the owner's manual for the appropriate peripheral device.
- *file-number*—A numeric expression specifying the number assigned to a file or device in an OPEN instruction.
- *row and column*—Numeric expressions specifying a screen character location. The screen can consist of as many as 24 horizontal rows and 40 vertical columns. Rows are numbered from 1 to 24, with row 1 being the top row and row 24 the bottom. Columns are numbered from 1 to 40, with column 1 being the far left column, and column 40 the far right.
- The number of rows and columns available to certain instructions can vary according to the current Graphics Mode and margin settings.

- *pixel-row and pixel-column*—Numeric expressions specifying a screen pixel location. The screen can consist of as many as 192 horizontal pixel-rows and 256 vertical pixel-columns. Pixel rows are numbered from 1 to 192, with pixel-row 1 being the top pixel-row and pixel-row 192 the bottom. Pixel-columns are numbered from 1 to 256, with pixel-column 1 being the far left pixel-column, and pixel-column 256 the far right.

You can actually specify a *pixel-row* as high as 256; however, only *pixel-rows* 1 through 192 are on the screen. It might be helpful to think of *pixel-rows* 193-256 as being "below" the screen. The number of *pixel-rows* and *pixel-columns* available to certain instructions may vary according to the current Graphics Mode.

- *character-code*—A numeric expression with a value from 0 to 255, specifying one of the 256 ASCII characters.
- *sprite-number*—A numeric expression with a value from 1 to 32, specifying the number assigned to a sprite with the SPRITE subprogram.
- *color*—A numeric expression with a value from 1 to 16, specifying one of the 16 available colors.