Texas Instruments Home Computer

Mini Memory

SOLID STATE SOFTWARE™ **COMMAND MODULE**

An expansion module that increases the memory capacity and power of your computer with these features:

- ***Adds 4K bytes of Random Access Memory (RAM) for program and data storage, 4K bytes of Read Only Memory (ROM), and 6K bytes of Graphics Read Only Memory (GROM).***
- ***Contains additional TI BASIC subprograms and utility routines that allow you to load and link BASIC programs to assembly language subprograms, to access the machine resources of your computer, and to address the Memory Expansion unit (sold separately) from a TI BASIC program.***
- ***Has a built-in battery to preserve data or programs stored in the module, even when you remove the module from the computer or turn the computer off.***
- ***Allows you to troubleshoot your assembly language programs by means of the EASY BUG debugging program.***
- ***Comes with a cassette-based Line-by-Line Symbolic Assembler that makes it possible for you to create your own assembly language programs.***

WARNING

To retain data in Mini Memory Module, turn console power off and discharge hand by touching some electrically grounded object before inserting or removing module.
STATIC DISCHARGE MAY CAUSE DATA LOSS!

1035987-2

TEXAS INSTRUMENTS
HOME COMPUTER

Mini Memory

This *Solid State Software*™ Command Module is designed to be used with the TI Home Computer. It increases the memory capacity and power of your computer with these features:

- Adds 4K bytes of Random Access Memory (RAM) for program or data storage in the module itself.
- Has a built-in battery to preserve data or programs stored in the module, even when you remove the module from the computer or turn the computer off.
- Contains 4K bytes of Read Only Memory (ROM) and 6K bytes of Graphics Read Only Memory (GROM) programmed with additional TI BASIC subprograms that allow you to link BASIC programs to assembly language subprograms, access the machine resources of your computer, and address the Memory Expansion unit (sold separately) from a TI BASIC program.
- Allows you to troubleshoot your assembly language programs by means of the EASY BUG debugging program.

TEXAS INSTRUMENTS
HOME COMPUTER

This program was adapted and developed by the staff of the Texas Instruments Personal Computer Division, with contributions by:

Wenn Lin
Granville Ott
Herman Schuurman
Paul Urbanus

This manual was developed by the staff of the Texas Instruments Learning Center, with contributions by:

Ira McComic
Jacquelyn Quiram
Jan Stevens

Mini Memory

TABLE OF CONTENTS

INTRODUCTION

The Mini Memory *Solid State Software*™ Command Module increases the versatility of your Texas Instruments TI-99/4 or TI-99/4A Home Computer by providing additional memory for your system and important tools for program development. In addition, the module contains a built-in battery, which permits the programs and data stored in the module's Random Access Memory (RAM) to be retained when the computer console is turned off, even if the module is removed from the console.

The features of the Mini Memory module include:

- A total of 14K bytes of memory. This memory consists of 6K bytes of Graphics Read Only Memory (GROM), 4K bytes of Read Only Memory (ROM), and 4K bytes of RAM. The programs resident in the GROM and ROM provide additional important program development tools. The RAM provides additional memory space for data and program storage.
- A built-in battery in the module to preserve the data or programs stored in the RAM memory.
- Additional files. Besides the 4K-byte RAM file in the Mini Memory module itself, the 24K-byte segment of the Memory Expansion unit, if attached, can be used by TI BASIC programs.
- Assembly language capabilities. With the Mini Memory module, assembly language object programs can be loaded into the module itself or into the Memory Expansion unit, if attached.
- Additional TI BASIC subprograms. With the Mini Memory module, several additional subprograms can be called with TI BASIC statements. These subprograms include the ability to PEEK and POKE values.
- Additional utility routines. The Mini Memory module includes several program routines which permit access to the computer's resources; for example, interfacing user programs with ROM- and GROM-resident programs, interfacing assembly language programs with the TI BASIC Interpreter, and accessing the Video Display Processor (VDP) RAM.
- A resident debug program. The EASY BUG debug program is a useful program-development tool with which you can access the internal resources of the computer system and troubleshoot your programs.

APPLICATIONS

You can use the RAM in the Mini Memory module to store either data or programs. This memory is "CPU memory," which means it is fast-access memory. You can take advantage of this fast storage and retrieval to store data which is used frequently in an application or to store assembly language programs which perform rapid computations.

A TI BASIC program which you frequently use can be stored in the Mini Memory module, rather than on a cassette tape or diskette, for quicker loading.

Generally, Random Access Memory (for example, the "user" memory in the computer console) loses its contents when the console is turned off. The battery-activated Mini Memory module, however, retains its contents when you turn the console off. Programs or data can be stored in the battery-powered RAM, the console turned off, and the module removed from the console. Then, when you reinsert the module and turn the console on, your data or program is ready to use.

CAUTION

> When you remove or insert the Mini Memory module, the computer console should be OFF to prevent the possibility of any data or programs stored in RAM being lost or altered.

An important feature of the Mini Memory module is its capability of implementing assembly language programs. The module allows you to load your own assembly language programs for direct access to the programmable components in the computer (such as the TMS9900 microprocessor or the TMS9918 Video Display Processor). Assembly language programs can also directly access devices such as the Wired Remote Controllers or cassette tape recorders through their interface ports on the console.

In addition, the module makes it possible for assembly language subroutines to be called from TI BASIC programs. These assembly language subroutines can perform functions which would be inefficient or impossible to implement in BASIC. Program routines resident in the module's GROM and ROM provide a convenient interface between TI BASIC programs and assembly language programs.

Also included in the module is the EASY BUG debugging program. With EASY BUG, you can access the memory and programmable components in the computer. EASY BUG also includes commands for loading and storing memory-image data on cassette tape.

GENERAL MEMORY INFORMATION

Note: In this manual, the greater than symbol (>) indicates that the following number is a hexadecimal (base 16) number.

The Mini Memory module contains a total of 14K bytes of memory, consisting of 4K bytes of battery-powered Random Access Memory (RAM), 4K bytes of Read Only Memory (ROM), and 6K bytes of Graphics Read Only Memory (GROM). Resident in ROM and GROM are a number of routines which add additional callable subprograms to TI BASIC and which are useful for interfacing assembly language programs with TI BASIC programs.

Appendix A shows the memory organization for the computer's entire memory space. The 4K bytes of ROM in the Mini Memory module occupy memory addresses >6000 through >6FFF (or from 24576 through 28671). Appendix B illustrates the details of the ROM organization.

The 4K bytes of RAM in the module occupy memory addresses >7000 through >7FFF (or from 28672 through 32767). Appendix C shows the details of the RAM organization when it is used for TI BASIC files. Appendix D describes how the RAM is organized when it is used for assembly language storage. The GROM occupies memory space which is not directly mapped into the CPU memory address space. Appendices E and F contain information about the RAM in the Video Display Processor (VDP).

SOURCES OF BACKGROUND INFORMATION

A number of references can give you helpful background information as you begin to use the Mini Memory module. These are:

- The Texas Instruments Home Computer *User's Reference Guide*;
- The Texas Instruments Home Computer *TI Extended BASIC* owner's manual;
- And, if you are creating assembly language programs, the Texas Instruments Home Computer *Editor/Assembler* owner's manual. This book (part number 1035984-1) is a complete reference guide to the TMS9900 assembly language and also contains details about the internal architecture of the TI-99/4 and TI-99/4A Home Computers. For information on ordering the *Editor/Assembler* manual, call toll-free 800-858-4565 (within the contiguous United States) or 800-692-4279 (within Texas), or write to:

  Consumer Relations Department
  Texas Instruments Incorporated
  P. O. Box 53
  Lubbock, Texas 79408

HOW TO USE THIS MANUAL

This manual assumes that you are already experienced in programming with TI BASIC. Statements, commands, and functions that are the same as in TI BASIC are only discussed briefly. For a complete description, see the *User's Reference Guide* included with your TI-99/4 or TI-99/4A Home Computer.

If you intend to use the Mini Memory module for creating your own assembly language programs, it is assumed that you are experienced in TMS9900 assembly language programming and that you are familiar with the internal organization of data and file structures used by the Home Computer. For a complete discussion of these topics, see the *Editor/Assembler* owner's manual.

The remainder of this manual explains the various features included in the Mini Memory module. The section entitled "Using the Module" explains the various options that are presented on the selection screen when the Mini Memory module is first brought into operation and explains how to select these options.

The section on "Loading and Saving TI BASIC Data Files" explains how to use the Mini Memory module for accessing data files, and the "Loading and Saving TI BASIC Programs" section describes how to use the module for loading and storing program files.

The section on "Additional TI BASIC Subprograms" explains the additional subprograms the Mini Memory module provides to interface with assembly language programs and the computer system.

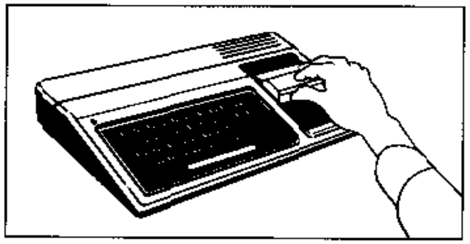
"Loading Assembly Language Programs" discusses the procedures for loading assembly language programs and subprograms, and the "System Utility Routines" section describes the Mini Memory module utilities which access CPU ROM and GROM routines.

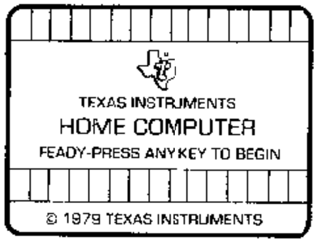
The "EASY BUG Debugger" section details the operation of the EASY BUG debugging program, and the six appendices contain information about the memory organization of the Home Computer.

USING THE MODULE

Before inserting or removing the Mini Memory module, it's a good practice to turn off the computer console. Turning the console off prevents the possibility of "contact bounce" between the module and console contacts, which could cause you to lose or alter the contents of the module's Random Access Memory.

Note: Be sure the module is free of static electricity before inserting it into the computer. (See "Service and Warranty Information" for details about static electricity.)





1. Slide the module into the slot on the console. Then turn the computer ON, and wait for the master title screen to appear.
2. Press any key to make the master selection list appear. Two options for the Mini Memory module appear on this list: EASY BUG and MINI MEMORY. If you select EASY BUG, the EASY BUG debugging program is brought into operation and its selection screen appears. (For detailed information on EASY BUG, see the section "EASY BUG Debugger.") If you select MINI MEMORY, you can choose to load files, run programs, or initialize the Mini Memory module's Random Access Memory. Press the key corresponding to the number beside the desired option.

Note: To remove the module, first return the computer to the master title screen by pressing **QUIT**. Then turn the computer console OFF, and remove the module from the slot.

If you select MINI MEMORY from the master selection list, the following selections are available.


```
      * MINI MEMORY *

PRESS:
1 TO LOAD AND RUN
2   RUN
3   RE-INITIALIZE

© 1981 TEXAS INSTRUMENTS
```

Three options are presented. To select an option, press the appropriate key for that option.

LOAD AND RUN loads assembled programs in tagged or compressed object format (stored on diskette) into memory and runs the programs. RUN executes programs previously loaded into memory, and RE-INITIALIZE re-initializes the Mini Memory module and prepares it for loading new programs. These three options are discussed in the following paragraphs.

LOAD AND RUN

The LOAD AND RUN option allows you to load and execute assembly language programs developed with the Editor/Assembler package and stored on a diskette. When you press 1 to select the LOAD AND RUN option, the prompt "FILE NAME?" appears. This file must contain an assembly language program in object format. Type the filename and then press **ENTER**; for example, typing

```
DSK1.DEMO
```

and pressing **ENTER** loads a file named DEMO from a diskette in Disk Drive 1.

After the file is loaded, the filename is erased from the screen. The computer is now ready to accept another filename. You may load as many files as you like until the memory is full. When you have loaded all your files, press **ENTER** (without entering a filename) to proceed.

The prompt "PROGRAM NAME?" appears next. The program name is any entry point in a program which is marked by a label DEFined in the DEF list of the program. Pressing **ENTER** without entering a program name creates an error condition.

RUN

If you have previously loaded an assembly language object program, choose the RUN option to run the program. Remember, a program loaded into the Mini Memory module is retained even if the console is turned off. Therefore, you can run this program without reloading it.

If you press **2** when the Mini Memory selection list is on the screen, the prompt "PROGRAM NAME?" is displayed. Enter the name of the program. The program name must be an entry point in a program which appears in the internal REF/DEF table. (For more information on the REF/DEF table, see "LOAD Subprogram") If you press **ENTER** without entering a program name, the computer locates and runs the program most recently executed.

RE-INITIALIZE

If you press **3** to select the RE-INITIALIZE option, the Mini Memory module's Random Access Memory is initialized to accept new files. Any programs or data stored in the Random Access Memory are lost.

When you choose this option, the screen goes blank momentarily, and then the selection screen reappears.

If the message MEMORY ALREADY INITIALIZED, HIT "PROC'D" TO CONFIRM is displayed, press **PROC'D** if you want to re-initialize memory. Re-initializing clears all existing program references from the memory and prepares for loading new programs. *Note*: Press **PROC'D** ONLY if you want to load a new program and the remaining memory space is too small to add the new program.

If you do not want to re-initialize, press any other key to return to the selection list without re-initializing memory.

LOADING AND SAVING TI BASIC DATA FILES

Probably the most common application for the Mini Memory module is fast, temporary data storage for use by TI BASIC programs. Since it retains its data even if power to the console is turned off, the Mini Memory module is useful for preserving small amounts of data.

You can establish a file of up to 4K bytes in the Mini Memory module alone. If the Memory Expansion unit is connected to the computer console, the Mini Memory module also allows you to access an additional file, EXPMEM2, located in the Memory Expansion unit. This file can have a length of up to 24K bytes.

ADDITIONAL FILES INTRODUCED TO THE SYSTEM

The Mini Memory module introduces two new files to the system.

1. MINIMEM—The 4K read/write memory segment located in the Mini Memory module itself.
2. EXPMEM2—A 24K memory segment located in the Memory Expansion unit.

The last file is available only if the Memory Expansion unit is connected to the system and turned on. Refer to the Memory Expansion unit owner's manual for more information on how to connect this unit to the system and the proper initialization procedure.

FILE ACCESS

The memory in the Mini Memory module and the Memory Expansion unit can be used for data file storage by TI BASIC at any time. However, if you want to use these files for data storage together with storing assembly language programs, you must take certain precautions to avoid destroying data and/or assembly language programs. See "Mixing Assembly Language Programs and TI BASIC Files" for more information on this procedure.

WARNING

If data files are stored in the the Mini Memory module (the file called MINIMEM), the assembly language capabilities cannot be used.

Data File Specifications

The following specifications define data files to be stored in the Mini Memory module.

- File Organizations—SEQUENTIAL and RELATIVE.
- File types—DISPLAY and INTERNAL.
- Record length—VARIABLE and FIXED.
- Operation modes—INPUT, OUTPUT, UPDATE, and APPEND.
- BASIC functions—EOF.

The following restrictions apply to the above specifications.

- The VARIABLE-length record type can be used only with SEQUENTIAL files.
- For a file with VARIABLE-length records, a zero-length data item in the first record will be stored incorrectly. To ensure proper file operation, make sure that the first record in your file is not a null string.

For more information on data file handling and accessing files, refer to the "File Processing" section of the *User's Reference Guide*.

Setting Up a Mini Memory File for Data Storage

You can think of the files introduced to the system by the Mini Memory module as high-speed, out-of-program storage files, just as a cassette or diskette is an out-of-program storage file. The TI BASIC statements used to set up and access files in the Mini Memory module are the same as those described in the *User's Reference Guide*.

To access a file, you must open it with an OPEN statement, listing the file specifications you want the file to have.

```
OPEN #3:"MINIMEM",RELATIVE,FIXED,UPDATE,DISPLAY
```

Data can be written to the file with a PRINT statement and read from the file with an INPUT statement. The RESTORE statement repositions the file at its beginning record.

```
PRINT #3: A,B,C,D
RESTORE #3
INPUT #3: A,B,C,D
```

You should close the file when you no longer need to access it or if you want to re-OPEN it to establish different specifications (like changing it from an OUTPUT file to an INPUT file).

```
CLOSE #3
```

Setting Up a Memory Expansion File for Data Storage

Setting up data files in the 24K-byte memory segment of the Memory Expansion unit requires the same procedure as that described for the Mini Memory module, with one exception.

To ensure that a file is properly opened and closed, each OPEN statement *must* be preceded by a CALL LOAD statement specifying an address and a value. (See "Additional TI BASIC Subprograms" for full information about the LOAD subprogram.) The address is the same for each CALL LOAD statement; the value that follows the address depends on the file type and record length.

For INTERNAL-type files with VARIABLE-length records, the format is

```
CALL LOAD(-24574,24)
OPEN #1:"EXPMEM2",SEQUENTIAL
,VARIABLE 32,INTERNAL,OUTPUT
```

For DISPLAY-type files with VARIABLE-length records, the format is

```
CALL LOAD(-24574,16)
OPEN #1:"EXPMEM2",SEQUENTIAL
,VARIABLE 32,DISPLAY,OUTPUT
```

For INTERNAL-type files with FIXED-length records, the format is

```
CALL LOAD(-24574,8)
OPEN #1:"EXPMEM2",RELATIVE,F
IXED,UPDATE,INTERNAL
```

For DISPLAY-type files with FIXED-length records, the format is

```
CALL LOAD(-24574,0)
OPEN #1:"EXPMEM2",RELATIVE,F
IXED,UPDATE,DISPLAY
```


Reading and Writing a Data File

The following programs illustrate writing data to the MINIMEM and EXPMEM2 files and then reading the data.

MINIMEM Example:

```
100 OPEN #5:"MINIMEM",SEQUEN
TIAL,FIXED,OUTPUT,INTERNAL
110 INPUT X
120 INPUT Y
130 INPUT Z
140 PRINT #5:X, Y, Z
150 CLOSE #5
```

This segment opens the MINIMEM file (the 4K bytes of RAM in the Mini Memory module) as an output file in statement number 100. Lines 110 through 130 accept the data values entered from the keyboard. Line 140 writes these values to the MINIMEM file, and line 150 closes the MINIMEM file.

At this point, the computer console can be turned off and the Mini Memory module removed from the console. The data is preserved just as if it had been stored on a cassette or diskette.

The following segment reads the data values stored in the MINIMEM file and displays the values on the screen.

```
200 OPEN #5:"MINIMEM",SEQUEN
TIAL,FIXED,INPUT,INTERNAL
210 INPUT #5: P, Q, R
220 PRINT P, Q, R
230 CLOSE #5
```

EXPMEM2 Example:

```
100 CALL CLEAR
110 REM OPEN FILE FOR DISPLA
Y-TYPE,VARIABLE-LENGTH
120 CALL LOAD(-24574,16)
130 OPEN #1:"EXPMEM2",SEQUEN
TIAL,VARIABLE,DISPLAY,UPDATE
140 FOR I=1 TO 20
150 PRINT #1:"RECORD #";I;"W
AS READ."
160 NEXT I
170 RESTORE #1
180 FOR J=1 TO 20
190 INPUT #1:A$
200 PRINT A$
210 NEXT J
220 CLOSE #1
```

This program opens a file in EXPMEM2 (the 24K-byte memory segment in the Memory Expansion unit), writes 20 records to the file, and then reads the records back and displays them on the screen. Note the CALL LOAD statement in line 120, which precedes the OPEN statement in line 130, and the RESTORE statement in line 170, which repositions file #1 at its beginning record.

Note: When the computer console is turned off, any data stored in the Memory Expansion unit is destroyed.

LOADING AND SAVING TI BASIC PROGRAMS

In addition to data file storage, the Mini Memory module is also useful for storing short TI BASIC programs or assembly language programs. Assembly language programs stored on diskette are loaded through the LOAD AND RUN option on the Mini Memory selection list, while TI BASIC programs can be saved and loaded using the SAVE and OLD commands, respectively. For more information on these commands, refer to the "Commands" section of the *User's Reference Guide*.

The Mini Memory module can store nearly 4K bytes (exactly 4088 bytes) of program data in its RAM.

To use files for assembly language programs and TI BASIC files together, you must take certain precautions, which are described in the section "Mixing Assembly Language Programs and TI BASIC Files."

LOADING AND STORING A PROGRAM

The following procedure shows you how to create a one-statement test program, save it in the Mini Memory module, and then load it back into the console's memory.

First, select TI BASIC and enter the program.

```
100 PRINT "THIS IS A TEST"
```

Store the program by entering the command

```
SAVE MINIMEM
```

After the program is stored in the module, the computer console can be turned off. Then, even if the Mini Memory module is removed from the console, the program is preserved just as if it had been stored on a cassette or diskette.

As a test, if you do not want to turn the console off at this point, enter the command

```
NEW
```

to remove the program from the console memory. To load the program back into the console memory, enter the command

```
OLD MINIMEM
```

Now, enter the LIST command to see that the program was loaded back into memory.

MIXING ASSEMBLY LANGUAGE PROGRAMS AND TI BASIC FILES

Assembly language programs and TI BASIC files *cannot* be stored simultaneously in the Mini Memory module. If the Mini Memory module and the Memory Expansion unit are both available, however, you can mix assembly language programs and TI BASIC files, with the following restrictions.

- The Mini Memory module must be used for assembly language storage only.
- You can also store assembly language programs in the 8K-byte segment of the Memory Expansion unit.

- The 24K-byte segment of the Memory Expansion unit must be used for your TI BASIC files.

WARNING

f data files are stored in the Min Memory module (the file called MINIMEM), the Memory Expansion unit cannot be used for assembly program storage.

When you have both the Mini Memory module and the Memory Expansion unit and you want to mix assembly language programs and TI BASIC files, use the following steps to avoid destroying data and/or programs.

1. Initialize the Mini Memory module by following one of two procedures. One procedure is to select the RE-INITIALIZE option from the Mini Memory selection screen. A second procedure is to select TI BASIC from the master selection list and use the CALL INIT command. (See "Additional TI BASIC Subprograms" for a description of the INIT subprogram.)
2. From TI BASIC, use the OPEN statement to reserve the EXPMEM2 file for data storage.
3. LOAD the assembly language programs you want to use. (See "Loading Assembly Language Programs" for additional information.)

ADDITIONAL TI BASIC SUBPROGRAMS

Several subprograms included in the Mini Memory module provide an interface between assembly language programs and TI BASIC. These subprograms are INIT, LOAD, LINK, PEEK, PEEKV, POKEV, and CHARPAT. Each subprogram is discussed in this section. In these discussions, the term "CPU memory" refers to all the memory directly accessible by the Central Processing Unit (CPU). This includes the memory in the module itself, the Memory Expansion unit, if attached, and the scratchpad memory in the console.

INIT SUBPROGRAM

Format: CALL INIT

The calling statement of the INIT subprogram has no arguments. We recommend that you generally use CALL INIT in the command or immediate mode to avoid inadvertently losing programs or data stored in memory. However, if the CALL INIT statement is used in a program, it must appear prior to the LOAD and LINK subprograms.

The INIT subprogram initializes the CPU memory for assembly language subroutines and re-initializes the internal tables in the Min Memory module. When this subprogram is run, it checks to see if the Memory Expansion unit is connected. If so, it sets the corresponding table values in the Mini Memory module to enable access to both the module and the Memory Expansion unit.

WARNING

CALL INIT erases all programs and data from the Mini Memory module. Use it only to clear the memory for loading new programs or subroutines. Also, if the Memory Expansion unit is not properly connected or if it is not turned on when CALL INIT is executed, the INIT subprogram does not recognize the Memory Expansion unit. If this happens, this memory cannot be used for loading programs.

Since the Mini Memory Command Module contains its own internal power supply, the module does not have to be initialized every time the main console is turned on. Only if you want to re-initialize the module's memory does the INIT subprogram have to be used.

CAUTION

The Mini Memory module retains only the data contained in the module itself. Any data in the Memory Expansion unit is lost if the system is turned off.

LOAD SUBPROGRAM

The LOAD subprogram serves two purposes.

- It loads assembly language object files into CPU memory.
- It loads data into CPU memory.

The syntax for the CALL LOAD statement has two forms, depending on the purpose of the CALL LOAD statement.

Loading Object Files

Format: CALL LOAD(*obj-filename*[,*obj-filename*,...])

This format of the CALL LOAD statement loads an assembly language object file or pokes direct data into memory for later execution by the CALL LINK statement.

The *obj-filename* (object filename) can be any valid string expression and specifies the file to be opened and read by the LOAD subprogram. Relocatable object code is loaded at the first available address, which depends upon the system configuration; and space is reserved for the assembly language programs according to the length specified in the "0-tag" field in the object file. (For a description of object program tag fields, see the *Editor/Assembler* owner's manual.) Absolute object code is loaded at the absolute address specified in the object code.

For example, the statement

```
CALL LOAD("DSK1.DEMO")
```

loads the file DEMO from the diskette in Disk Drive 1.

CAUTION

Absolute code is loaded at the address specified in the object code. No space is reserved unless the length is specified in the "0-tag" field. Loading data into memory used by the TI BASIC interpreter can cause the system to crash.

If you are using the Mini Memory module only, ***without*** the Memory Expansion unit attached to the console and turned on, the first assembly language program is loaded starting at >7118, the lowest available address in the module's Random Access Memory (RAM). If the Memory Expansion unit is connected and turned on, the first assembly language program is loaded starting at >A000, the starting address of the highest memory segment in the Memory Expansion unit. Subsequent programs are loaded sequentially, beginning with the lowest address in the high memory area.

See "Loading Assembly Language Programs" for additional information.

Loading Data Files

Format: CALL LOAD(*address,value*[,...,"",*address,value*,...])

When the LOAD subprogram is used to load data into CPU memory, a list of integers (called a poke list) should be specified. The poke list should start with an address between 0 (>0000) and 32767 (>7FFF) or an address between −1 (>FFFF) and −32768 (>8000), followed by a list of integers to be used as one-byte data values. These integer values are loaded into consecutive locations, starting at the given address. An empty string ("") separates the last byte of one poke list and the starting address of the next. The address for a poke list is absolute, and the data is not-relocatable.

For example, the statement

```
CALL LOAD (-32000,255,21,"",8197,85)
```

loads the value >FF15 at memory word address >8300 (byte addresses >8300 and >8301) and the value >55 at memory byte address >2005.

If an object code program is loaded directly with a poke list, a name entry must also be loaded so that the program can be accessed by a CALL LINK statement (described below). The program name and address are added to the REF/DEF table in the module's memory in the following manner.

First, the First Free Address in the module (FFAM) and the Last Free Address in the module (LFAM) must be read from memory by means of the PEEK command (described below). The addresses of these two variables are >701C and >701E respectively. After checking that there is enough room (8 bytes) to add another label to the REF/DEF table, subtract 8 from the old LFAM, and poke the new LFAM value to >701E using the CALL LOAD statement. Load the program name (must be exactly 6 bytes, including spaces) and the program address (2 bytes) into the 8-byte space added to the REF/DEF table.

For example, if the LFAM is >8000, change it to >7FF8 and load the name and then the address of the program.

LINK SUBPROGRAM

Format: CALL LINK(*program-name* [,*parameter-list*,"",...])

The LINK subprogram passes control and, optionally, a list of parameters from a TI BASIC program to an assembly language program.

The *program-name* is a string expression consisting of from one through six characters and must be an entry in the REF/DEF table. This name must be defined in a program which has been loaded previously. Or, if a program was loaded byte-by-byte by means of a poke list in a CALL LOAD statement, the program name must have been entered in the REF/DEF table explicitly. See the "LOAD Subprogram" section for more information.

The *parameter-list* is optional. This list is used when parameters need to be passed between the assembly language program and TI BASIC. You can pass string or numeric variables or expressions.

Parameter Passing

Depending upon whether a parameter is a variable or an expression, the parameter is passed by name or by value. Variables are passed by name, and expressions are passed by value.

If a variable is passed to an assembly language program, it can have its value changed in the assembly language program, thus changing the value of the variable in the main program also. If the variable in a parameter list has not appeared in previous TI BASIC statements, the interpreter creates a Symbol Table entry for the variable.

Expressions are passed by value, since they are not directly associated with a variable. The value of an expression cannot be passed back to the calling program.

When an array element, such as A(9), is given in the parameter list, it is passed as a variable. An entire array can be passed by following the parameter name with parentheses. If the array has more than one dimension, a comma must be placed inside the parentheses for each additional dimension. For example, A( ) indicates a one-dimensional numeric array called A. EXT$(,,) represents a three-dimensional string array called EXT$.

To specify that certain variables are to be used only for passing a value, but not for returning results, the variable can be enclosed in parentheses. For example, (SUMI) refers to the current value of the numeric variable SUMI. (A$(5)) refers to the value of the string array element A$(5). Notice that complete arrays cannot be passed by value but must be passed by name; thus, (A( )) would be illegal.

A maximum of 15 arguments can be listed in the parameter list.

Operation

The LINK subprogram performs the following actions.

- It evaluates the assembly language program name and its length (1 to 6 characters) and pushes this information onto the value stack.
- It builds the argument list, consisting of identifiers for each argument in the parameter list, and builds a stack entry for each argument.
- It moves the program name to the area where the utility routine can access it and transfers control to the utility program.
- Upon return, it branches to an error routine if an error has been detected. Otherwise, it clears the stack entry used during LINK execution and returns to the TI BASIC calling program.

The LINK subprogram passes information about the arguments via the argument-identifier list in CPU RAM and the value stack in VDP RAM.

The argument identifiers, old value stack pointer, and the number of arguments in the list are located in the following CPU RAM locations.

Address	*Contents*
>7002->7011	Argument identifier, one byte for each argument.
>8310	Old value stack pointer of TI BASIC interpreter.
>8312	Number of arguments in the parameter list.

The argument-identifier codes are as follows.

0	Numeric expression
1	String expression
2	Numeric variable
3	String variable
4	Numeric array
5	String array

More information on each argument is stored in an eight-byte value stack in VDP memory. The structure of an individual value stack depends upon the type of argument, as described below.

Numeric Expression—The stack contains the value of the numeric expression. The value is expressed in radix 100 notation. The first byte is the exponent of 100. If the exponent is positive, it is in excess of 64. A negative exponent is expressed as a value less than 64 in the first byte. The absolute value of the exponent is the difference between this value and 64. The other seven bytes contain 0 to 99, for radix 100 digits. If the number is negative, the first word (two bytes) is the two's complement of the number. For example,

>3F, >22, >00, >00, >00, >00, >00, >00 equals 0.34
>BE, >FB, >00, >00, >00, >00, >00, >00 equals −500

String Expression—A string stack entry consists of the following information.

Bytes 0-1	>001C.
Byte 2	>65 (The string tag used by the TI BASIC interpreter).
Bytes 4-5	The pointer to the value of the string in VDP memory.
Bytes 6-7	The length of the string. Byte 6 should always be zero since the maximum string length is 255 characters.

Numeric Variable—This item is either a numeric variable or a numeric array element. The stack contains the following information.

Bytes 0-1	The pointer to the variable's Symbol Table entry in VDP memory.
Byte 2	Zero.
Bytes 4-5	The pointer to the eight-byte value of the variable in VDP memory.

String Variable—This item is either a string variable or a string array element. The stack entry contains the following information.

Bytes 0-1	The pointer to the variable's Symbol Table entry in VDP RAM.
Byte 2	>65 (The string tag used by the TI BASIC interpreter).
Bytes 4-5	The pointer to the string's value in VDP memory.
Bytes 6-7	The string length.

Numeric Array—This entry results from an argument of the form A( ), A(,), etc. It is used to permit a subprogram to manipulate an entire array. The string entry contains the following information.

Bytes 0-1	The pointer to the array's Symbol Table entry in VDP RAM.
Byte 2	Zero.
Bytes 4-5	The pointer to the array's value space in VDP RAM. The value space for a numeric array has two bytes for each dimension which indicate the maximum index for that dimension. The values of the rest of the elements are stored in sequential order.

String Array—This entry is similar to the entry for a numeric array, except that byte 2 contains >65. The value space for a string array contains two bytes for each dimension, indicating the maximum index, followed by a pointer to each array element's value (string value) in VDP RAM. Note that, with a numeric array, each array element is stored consecutively in the same memory area, while the elements of a string array are located in non-contiguous memory areas.

Name Link Routine

When an asssembly language subroutine is called from TI BASIC by a CALL LINK statement, control passes to the subroutine through a Name Link routine which resides in the utility program. The Name Link routine finds the name of the routine in the REF/DEF table located at the high end of the Mini Memory module's memory. When an assembly language program is loaded, the Loader adds an eight-byte entry to the REF/DEF table when it sees a REFed or DEFed label. This REF/DEF table starts at >7FFF and goes down toward >7118, the First Free Address (FFAM) in the module.

The REF/DEF table is searched from the lowest address up. Therefore, if two routines are loaded with the same name, the second one loaded is used. If the name you supply is greater than six characters or if the Name Link routine cannot find the name in the table, an error results.

The Name Link routine transfers control to the assembly language program with a 9900 branch-and-link instruction (BL). When the assembly language program is called from the link routine, the workspace is located at >70B8, and the return address is in R11 of that workspace. Before returning, your program should clear the byte at >837C; otherwise, an error message may be displayed, even though the program did not generate an error.

The assembly language program can assign new values to numeric or string variables or to elements of numeric or string arrays with utilities provided by the system. These utilities are described in the "System Utility Routines" section.

Entries on the value stack which result from parameters passed by the CALL LINK statement are automatically cleared by the LINK subprogram. If you directly manipulate the value stack, however, you must restore the stack to its original state before returning control to the LINK subprogram.

PEEK SUBPROGRAM

Format: CALL PEEK(*address, var*[,*var*,...,"",*address, var*,...])

The PEEK subprogram is used to read bytes of CPU RAM directly into TI BASIC variables.

The *address* parameter must be either a numeric expression or a numeric variable. The *address* is a decimal value from −32768 to 32767, representing a two-byte integer value. Addresses above >7FFF are written as negative numbers, treating the value as a two's-complement integer. (For example, to access an address above 32767, subtract 65536 from it.)

The variable list (*var* parameters) must consist of numeric variables. Each consecutive byte read from the memory is assigned to each variable in the order listed in the variable list. A null string ("") separates one PEEK sequence from the next one so that you can repeatedly PEEK several locations of memory in a single statement.

For example, the statement

```
CALL PEEK(8192,A,B,C(8),"",24576,X)
```

reads three bytes from address >2000 and up; assigns the values to the variables A, B, and C(8), consecutively; reads one byte from location >A000; and stores the value in variable X.

The returned value is a one-byte value and is always in the range of 0 through 255.

PEEKV SUBPROGRAM

Format: CALL PEEKV(*address, var*[,*var*,...,"",*address, var*,...])

The PEEKV subprogram is used to read bytes from VDP RAM. It works exactly as PEEK does, except that PEEKV accesses VDP RAM instead of CPU RAM.

The *address* is a decimal value from 0 to 16383, and the variable list (*var* parameters) is a list of numeric variables which are to contain the values read. Note that the VDP has 16K of RAM, and trying to access a memory address higher than 16383 may crash the system. Also, see "PEEK Subprogram" for more information.

POKEV SUBPROGRAM

Format: CALL POKEV(*address, var*[,*var*,...,"",*address, var*,...])

The POKEV Subprogram allows you to modify the value in the VDP RAM. It works the same way as LOAD works when LOAD is used to modify CPU memory.

The *address* is a decimal value from 0 to 16383 and *var* is a numeric expression or numeric variable that contains a value to be placed in the VDP memory at the specified address. Each specified value is stored consecutively beginning at the given address. For example, the statement

```
CALL POKEV(784,30,30,30,"",2,V)
```

changes color table 16 to color table 18 (>310 to >312 in the VDP RAM), resulting in a black foreground and gray background. If the value of V is 161, the character "A" appears in the top left corner of the screen.

CHARPAT SUBPROGRAM

Format: CALL CHARPAT(*char-code, str-var*[,*char-code,str-var*,...])

The CHARPAT subprogram returns a 16-character pattern identifier that specifies the pattern of the character-code

The *char-code* (character code) is any character number ranging from 32 through 159. Character codes 32 through 95 (through 127 on the TI-99/4A) are normally reserved for ASCII characters and are initially defined by the TI BASIC interpreter. The string expression (definition) of the character code is read into the *str-var* (string variable). This expression consists of 16 characters of hexadecimal digits that represent the character. Refer to the CHAR subprogram discussion in the *User's Reference Guide* for more details on defining a character.

LOADING ASSEMBLY LANGUAGE PROGRAMS

The Mini Memory module and the Memory Expansion unit are a powerful team. However, when they are used together, some care must be exercised to ensure that your relocatable object files are loaded into the proper memory areas.

If both the Mini Memory module and the Memory Expansion unit are being used, relocatable programs are loaded into the memory space in the following sequence.

1. The Memory Expansion unit's highest memory segment (the 24K-byte area starting at >A000).
2. The Memory Expansion unit's lowest memory segment (the 8K-byte area starting at >2000).
3. The Mini Memory module's memory (the 4K-byte area starting at >7118).

The first free address in high memory is initialized to >A000 by the INIT subprogram, and relocatable code is realocated to the starting load address. Whenever a "0-tag" is encountered, the starting load address is updated from the first free address in high memory, and the program length is added to this address. Subsequent programs are loaded sequentially, beginning with the lowest address in the high memory area. (See Appendix A for a map of CPU memory when both the Mini Memory module and the Memory Expansion unit are utilized.)

If you are using the Mini Memory module only, *without* the Memory Expansion unit attached to the console and turned on, your program loads directly into the module's Random Access Memory (RAM). The first assembly language program is loaded starting at >7118, the lowest available address in the module's RAM.

At times, you may want to load a program directly into the Mini Memory module when the Memory Expansion unit is attached, bypassing the normal loading sequence. To do so, it is necessary to make the Memory Expansion unit temporarily "invisible" to the system by clearing the values in memory locations >7022 through >7029 (see Table 2, below). These are the pointer values that indicate the presence of the Memory Expansion unit (see Table 1 for these values).

The easiest way to accomplish this task is to use a short TI BASIC program including two versions of the LOAD subprogram, one with a "poke" list and one that loads the assembly language program into the Mini Memory module.

```
CALL INIT
100 CALL LOAD(28706,0,0,0,0,
0,0,0,0)
110 CALL LOAD("DSK1.DEMO")
120 CALL LINK("LINES")
```


The CALL INIT command initializes the system, clearing any previously loaded data or program pointers. The first line of the program zeroes references to the Memory Expansion unit, starting at memory location 28706 (>7022). Line 110 loads a program named DEMO from a diskette in Disk Drive 1, and line 120 executes DEMO, starting at the entry point labeled LINES.

If you want to re-inform the system of the existence of the Memory Expansion unit, you can again use CALL LOAD with a poke list of the appropriate decimal values (see Table 1).

Table 1. Mini Memory variables, with Memory Expansion unit attached and turned on.

Location	*Hex Value*	*Decimal Value*
>7022	>A0	160
>7023	>00	0
>7024	>FF	255
>7025	>E0	224
>7026	>20	32
>7027	>00	0
>7028	>3F	63
>7029	>FF	255

Table 2. Mini Memory variables, with Memory Expansion unit unattached, turned off, or "invisible."

Location	*Hex Value*	*Decimal Value*
>7022	>00	0
>7023	>00	0
>7024	>00	0
>7025	>00	0
>7026	>00	0
>7027	>00	0
>7028	>00	0
>7029	>00	0

You can also use the M (Modify) command in EASY BUG to restore the table values so that the system again recognizes the presence of the Memory Expansion unit.

Note: When you are creating assembly language programs, it is important to know how to use the proper assembly language directives to make sure programs and associated data are loaded correctly. Refer to the *Editor/Assembler* owner's manual for guidelines on writing a program so that it will load properly.

SYSTEM UTILITY ROUTINES

The utility routines resident in the Mini Memory module can be called from an assembly language program to access machine resources and interface with the TI BASIC interpreter. The use of these routines requires a knowledge of the routines themselves and the organization of data used by the routines. Additional information on these topics is included in the *Editor/Assembler* owner's manual.

Two types of utility programs are provided in the Mini Memory module.

- One program contains a collection of standard system utilities with which to link to ROM/GROM routines, perform a keyboard scan, access the VDP, etc.
- A second program contains TI BASIC interface utilities with which an assembly language program can access variables passed through a CALL LINK statement in a TI BASIC program. This program also contains an error-handling utility to return exceptions to a TI BASIC program.

STANDARD UTILITIES

All utility routines use UTILWS (address >7092) for utility workspace registers, and all parameters are passed through the calling program's workspace registers. For your convenience, USRWSP (address >70B8) is reserved for your program's set of workspace registers. However, any register area you provide can be used to pass parameters.

The following sections describe the data-passing conventions and calling-statement syntax for each routine.

VDP Single Byte Write — VSBW

Format: BLWP @VSBW Equates VSBW to >6024.

This routine writes a single-byte value to a specified VDP RAM address.

R0	The VDP RAM address.
R1	A one-byte value in the most significant byte of Register 1.

Example:

```
       LI    R0,>0200    VDP RAM address >0200.
       LI    R1,>4100    Character code for A.
       BLWP  @VSBW       Display the character.
```

This program displays the character "A" on the screen at location >0200.

VDP Multiple Byte Write — VMBW

Format: BLWP@VMBW Equates VMBW to >6028.

This routine writes multiple bytes from CPU RAM to VDP RAM.

R0	VDP RAM address.
R1	Starting address of CPU RAM buffer.
R2	Number of bytes to write.

Example:

```
       LI    R0,>018E    VDP RAM address >018E.
       LI    R1,HI       Address of text.
       LI    R2,5        Number of bytes to write.
       BLWP  @VMBW       Display the characters.
       ..
       ..
HI     TEXT  'HELLO'     Text to be displayed.
```

This program displays the word "HELLO" in the middle of the screen (VDP RAM address >018E).

VDP Single Byte Read — VSBR

Format: BLWP@VSBR Equates VSBR to >602C.

This routine reads a single byte from a specified VDP RAM address.

R0	VDP RAM address.
R1	The value read from VDP RAM in the most significant byte.

Example:

```
        LI      R0,>0380    VDP RAM address >0380.
        BLWP    @VSBR       Read one byte.
```

This program reads one byte of the color table (>0380) into the most significant byte of Register 1.

VDP Multiple Byte Read — VMBR

Format: BLWP@VMBR Equates VMBR to >6030.

This routine reads multiple bytes from VDP RAM into CPU RAM.

R0	VDP RAM address to read from.
R1	Starting address of CPU RAM buffer.
R2	Number of bytes to read.

Example:

```
        LI      R0,>0300    VDP RAM address >0300
        LI      R1,BUFF     Buffer area.
        LI      R2,>0080    Load number of bytes to read.
        BLWP    @VMBR       Read the bytes.
        ...
        ...
BUFF    BSS     >0080       Set up buffer for bytes read.
```

This program copies the >0080 bytes of sprite attribute list data from VDP RAM >0300 into the buffer area called BUFF.

VDP Write to Register — VWTR

Format: BLWP@VWTR Equates VWTR to >6034.

This routine writes a single-byte value to any of the VDP RAM registers.

R0	Least significant byte contains the value to be written; most significant byte contains the VDP register number (0 through 7).

Keyboard Scan — KSCAN

Format: BLWP@KSCAN Equates KSCAN to >6020.

This routine scans a specified keyboard and returns a keycode and status. The following memory locations are used for communication between a user program and the routine.

>8374	Keyboard device number. This one-byte number must be specified by your program. The meaning of this byte is the same as the key-unit in the TI BASIC KEY subprogram. See the *User's Reference Guide* for more information on the KEY subprogram.
>8375	ASCII value of the key pressed (one byte).
>8376	Wired Remote Controller Y-position (one byte).
>8377	Wired Remote Controller X-position (one byte).
>837C	GPL status register (one byte).

The GPL (Graphic Programming Language) status byte may be tested on return before the keycode is read. You can do this with a Compare Ones Corresponding (COC) instruction. (See the *Editor/Assembler* owner's manual for more information on this instruction.) Bit 5 of the GPL status byte is set if a key was pressed on the last call to KSCAN. The GPL status bits are assigned as follows.

H	GT	COND	CARRY	OVF	0	0	0
7	6	5	4	3	2	1	0

See the "Extended Utilities" section for a more complete description of the other status bits.

EXTENDED UTILITIES

Extended utilities are provided to access routines in the console GROMs and ROMs. These utilities are GPLLNK (link to GPL routines in GROM), XMLLNK (link to routines in ROM), and DSRLNK (link to Device Service Routines).

Since the extended utilities access routines in the console, extreme care should be taken when you use these utilities. You should make sure that the GPL workspace registers are not changed, the memory space used by the console routine is set up properly, and the routine returns correctly to your program.

Linking to GROM-Resident Routines — GPLLNK

Format:	BLWP@GPLLNK	Equates GPLLNK to >6018.
	DATA *console-routine-address*	Gives address of GPL routine to be executed.

The GPLLNK routine sets an internal flag to indicate that a GPL program has been called from an assembly language program, loads the GPL workspace (address >83E0), branches to the GROM code, and executes the GPL routine specified by the DATA directive.

The GPL routine must return with a RTN statement in order for the program to transfer back to the caller. When the RTN statement is encountered in the GPL routine, the statement returns to the system routine. The system routine checks the internal flag and, finding it set, returns further back to the assembly language routine.

Some of the addresses of GPL routines and their calling and returning conventions are given below. The names FAC, STACK, and STATUS are used in the following descriptions. FAC is equated to >834A, STACK is equated to >836E, and STATUS is equated to >837C. STATUS is the GPL status byte. It is organized as follows.

	High	Greater	Condition	Carry	Overflow	Unused
Bit	7	6	5	4	3	2,1,0

Bit 7 High bit. Controlled during the execution of the GPL interpreter.

Bit 6 Greater than bit. Controlled by the GPL interpreter during the execution of the GPL program.

Bit 5 Condition bit. Controlled by the GPL interpreter. The key-scan routine turns this bit on when a new key is detected. Also, the DSR routine turns this bit on to indicate that a file does not exist.

Bit 4 Carry bit. Controlled by the GPL interpreter.

Bit 3 Overflow bit. Controlled by the GPL interpreter.

The DATA directive specifies the address of the GPL routine to be executed. Each routine is described below.

DATA >0016 Load Standard Character Set—Loads the standard character set into VDP RAM.
- Input: FAC—Pointer to the beginning address in VDP RAM where characters are loaded.
- Output: VDP RAM at the address specified in FAC contains the standard character set.

DATA >0018 Load Small Character Set—Loads the small character set (for TEXT mode) into VDP RAM.
- Input: Same as Load Standard Character Set.
- Output: Same as Load Standard Character Set.

DATA >0020 Execute Power-Up Routine—Powers up and initializes the system.
- Input: None.
- Output: The system is powered up and initialized. The sound and VDP circuits are cleared; the default values for the VDP registers, character set, color table, and status block are loaded. The available VDP RAM size is stored at >8370.

DATA >0034 Accept Tone—Issues an accepting tone for input. No memory setup is required prior to calling the routine.

DATA >0036 Bad Response Tone—Issues a bad-response tone warning. No memory setup is required prior to calling the routine.

DATA >0038 Get String Space Routine—Allocates a memory space in VDP RAM with a specified number of bytes. This routine should not be used outside the TI BASIC environment. If there is not enough space, the routine does a "garbage collection" to eliminate temporary strings and then tries again. If there is still not enough space, the routine issues the MEMORY FULL error message.

Input: Addresses >830C and >830D should contain the number of bytes to be allocated.

Output: Address >831C points to the allocated string space and address >831A points to the first free address in VDP RAM. The four bytes at addresses >8356 through >8359 are used by this routine. The FAC area may be destroyed if a garbage collection is done.

Note: Although this routine is designed to allocate a string space in VDP RAM, it is also useful for assigning space for the Peripheral Access Block (PAB) and data buffer required by a DSR. See the *Editor/Assembler* owner's manual for a description of Peripheral Access Blocks.

DATA >003B Bit Reversal Routine—Provides a mirror image of a byte of information. It is used most commonly to form a mirror image of a character definition.

Input: FAC—Address of data in VDP RAM.
FAC+2—Number of bytes to reverse.

Output: The specified number of bytes in the VDP RAM are bit-reversed; that is, bits 0 and 7, bits 1 and 6, bits 2 and 5, and bits 3 and 4 are exchanged.

Side Effects: CPU RAM from >8300 through >8340 is erased.

DATA >003D Cassette Device Service Routine—Accesses the cassette DSR routine.

Input: The Peripheral Access Block and data buffer must be set up in VDP RAM prior to the call. The screen offset is >60 for TI BASIC and >00 outside the TI BASIC environment. The screen start address must be >00 for the prompts issued by the cassette DSR. FAC is the device name (for example, "CS1"). Address >8356 points to the first character *after* the name in the PAB. Addresses >8354 and >8355 are the length of the name (for example, >0003 for "CS1"). The word at address >83D0 should be set to >0000. Address >836D must be set to >08 to indicate a DSR call. The STATUS byte must be >00.

Output: The cassette DSR prompts for the operation of the cassette.

DATA >004A Load Lower Case Character Set—Loads the lower-case character set into VDP RAM. Input and Output are the same as those in loading other character sets.

Note: This routine applies only to the TI-99/4A console.

One of the uses for the GPL link routine is to call the floating-point routines written in GPL from an assembly language program. When these floating-point routines are called, the contents of CPU RAM locations >834A through >836F may be used, and VDP RAM locations >03C0 through >03DF are used as a buffer area.

The GPL status byte reflects the condition of the calculation. All of the input and output data values are in floating-point format.

When errors occur during the execution of floating-point routines, they are indicated in CPU RAM location >8354. The error codes are given below.

Code	*Error Description*
01	Overflow error.
02	Syntax error.
03	Integer overflow on conversion.
04	Square root of negative number.
05	Negative number raised to non-integer power.
06	Log of negative number or zero.
07	Invalid argument in trig function.

The floating-point routines are described below.

DATA >0014 Convert Number to String (CNS)—Converts a floating-point number to an ASCII string.

- Input: FAC—8-byte floating-point value.
  FAC + 11—If set to zero, the output string is in BASIC format. Otherwise, the output is in FIX mode, which requires data in FAC + 12 and FAC + 13.
  FAC + 12—If one, it expresses overflow from the calculation range by ± EE...E. Underflow is expressed as zero.
  FAC + 13—The number of digits to fix to the right of the decimal point. A negative value disables the FIX mode.
- Output: FAC—Modified.
  FAC + 11—The least significant byte of the address where the result string is located. The value >8300 must be added to obtain the real address.
  FAC + 12—The length of the string in bytes.

DATA >0022 Greatest Integer Function (INT)—Computes the greatest integer contained in the value.

- Input: FAC—The floating-point value.
- Output: FAC—The result. For positive numbers, the integer is the truncated value. For negative numbers, the integer is the truncated value plus one.
  STATUS—Set according to result.

DATA >0024 Involution Routine (PWR)—Raises a number to a specified power.

- Input: FAC—The exponent value.
  STACK—The pointer to the stack in VDP RAM which contains the 8-byte value.
- Output: FAC—The result in floating-point format. This is computed as EXP (exponent value) * LOG (ABS(base)).
  STATUS—Set according to result. Error conditions: Negative number raised to a non-integer power, and zero raised to a negative power.
- Side Effects: Locations >8375 and >8376 are destroyed, and the one-word content of >836E is decremented by 8. Also, the addresses FAC + 12 through FAC + 19 are destroyed.

DATA >0026 Square Root Routine (SQR)—Computes the square root of a number.

- Input: FAC—The input value.
- Output: FAC—The square root of the input value.
  STATUS—Set according to result.
- Side Effects: Addresses >8375 and >8376 are destroyed.

DATA >0028 Exponent Routine (EXP)—Computes the inverse natural logarithm of the input value.

Input:	FAC—The input value.
Output:	FAC—The resulting value. STATUS—Set according to result.
Side Effects:	Addresses >8375 and >8376 are destroyed.

DATA >002A Natural Logarithm Routine (LOG)—Computes the natural log of a number.

Input:	FAC—The input value.
Output:	FAC—The natural log of the input value. STATUS — Set according to result.
Side Effects:	Addresses >8375 and >8376 are destroyed.

DATA >002C Cosine Routine (COS)—Computes the cosine of a number.

Input:	FAC—The input value.
Output:	FAC—The cosine of the input value. STATUS—Set according to result.
Side Effects:	Locations >8375 and >8376 are destroyed.

DATA >002E Sine Routine (SIN)—Computes the sine of a number.

Input:	FAC—The input value.
Output:	FAC—The sine of the input value. STATUS—Set according to result.
Side Effects:	Locations >8375 and >8376 are destroyed.

DATA >0030 Tangent Routine (TAN)—Computes the tangent of a number.

Input:	FAC—The input value.
Output:	FAC—The tangent of the input value. STATUS—Set according to result.
Side Effects:	Locations >8375 and >8376 are destroyed.

DATA >0032 Arctangent Routine (ATN)—Computes the arctangent of a number.

Input:	FAC—The input value.
Output:	FAC—The arctangent of the input value. STATUS—Set according to result.
Side Effects:	Locations >8375 and >8376 are destroyed.

Before calling a GPL routine, check to see if any memory used by your program is accessed and modified in the routine. (Refer to the "Side Effects" described above.) Since CPU RAM is used by many system programs, it's easy to overlook information stored there. Also, some of these routines use up to 26 bytes of the BASIC Interpreter rollout area beginning at location >03C0 in VDP RAM. Therefore, exercise caution when you call a GPL routine.

Linking to ROM-Resident Routines — XMLLNK

Format:	BLWP @ XMLLNK	Equates XMLLNK to >601C.
	DATA *console-routine-code*	Defines ID code of ROM routine to be executed.
	or	
	DATA *console-routine-address*	Gives address of ROM routine to be executed.

Routines in the console ROM can be accessed through the XMLLNK routine. You can access a routine in console ROM in one of two ways.

One way is to specify the routine's code in a DATA statement. The low byte of the DATA statement should be set to zero. For example,

```
BLWP @XMLLNK
DATA >1200
```

branches to the routine CFI (convert floating point to integer) in the console.

A list of XML routine codes that can be called from an assembly language program are given in the following table.

Routine Code	*Name*	*Function*
06	FADD	Floating-point addition
07	FSUB	Floating-point subtraction
08	FMULT	Floating-point multiplication
09	FDIV	Floating-point division
0A	FCOMP	Floating-point compare operation
0B	SADD	Floating-point stack addition
0C	SSUB	Floating-point stack subtraction
0D	SMULT	Floating-point stack multiplication
0E	SDIV	Floating-point stack division
0F	SCOMP	Floating-point stack compare
10	CSN	Convert string to number
12	CFI	Convert floating-point format to integer
17	VPUSHG	Push a value into value stack
18	VPOP	Pop a value from value stack
23	CIF	Convert integer to floating point

The XML code, which is a single byte, is split into a high nybble, containing the XML table address, and a low nybble, containing the index into that table. There are 16 table addresses defined in the CPU address space. The high nybble specifies from which of the 16 tables to get the branch address, and the low nybble determines which of the 16 addresses in the table is to be used. Each table can contain up to 16 two-byte entry point addresses.

Another way to access a routine in the console ROM is to specify its address in the DATA statement. Note that the high bit of the DATA word must be set so that the system program recognizes this data as an address instead of an XML code. For example,

```
BLWP @XMLLNK
DATA >8D3A
```

branches to the console ROM address >0D3A, which is a floating-point compare routine.

WARNING

Using direct memory addresses of console ROM routines makes the assembly language program calling the routine completely machine dependent. Since the memory addresses of console ROM routines may change with future modifications, use of this method of access should be restricted to cases where there is no other reasonable way to achieve the required result.

FAC (the Floating Point Accumulator) starts at address >834A, ARG (which contains arguments) starts at address >835C, and STACK is at address >836E. The STATUS byte is at address >837C. All overflow errors, except in Convert Floating Point to Integer (CFI), return >01 at address >8354.

DATA >0600 Floating Point Addition (FADD)—Adds two values.
Input: FAC—First value.
ARG—Second value.
Output: FAC—Result of the addition.

DATA >0700 Floating Point Subtraction (FSUB)—Subtracts two values.
Input: FAC—Value to be subtracted.
ARG—Value from which FAC is subtracted.
Output: FAC—Result of the subtraction.

DATA >0800 Floating Point Multiplication (FMULT)—Multiplies two values.

Input: FAC—Multiplier.
ARG—Multiplicand.

Output: FAC—Result of the multiplication.

DATA >0900 Floating Point Division (FDIV)—Divides two values.

Input: FAC—Divisor.
ARG—Dividend.

Output: FAC—Result of the division.

DATA >0A00 Floating Point Compare (FCOM)—Compares two floating-point numbers.

Input: ARG—First argument.
FAC—Second argument.

Output: STATUS—Set according to result. The high bit is set if ARG is logically higher than FAC. The greater than bit is set if ARG is arithmetically greater than FAC. The equal bit is set if ARG and FAC are equal.

DATA >0B00 Value Stack Addition (SADD)—Subtracts using a stack in VDP RAM.

Input: STACK—Address in VDP RAM where the left-hand term is located.
FAC—Right-hand value.

Output: FAC—Result of the addition.

DATA >0C00 Value Stack Subtraction (SSUB)—Subtracts using a stack in VDP RAM.

Input: STACK—VDP RAM address that contains the left-hand term.
FAC—Value to be subtracted.

Output: FAC—Result of the subtraction.

DATA >0D00 Value Stack Multiplication (SMULT)—Multiplies using a stack in VDP RAM.

Input: STACK—VDP RAM address that contains the multiplicand.
FAC—Multiplier value.

Output: FAC—Result of the multiplication.

DATA >0E00 Value Stack Division (SDIV)—Divides using a stack in VDP RAM.

Input: STACK—VDP RAM address that contains the dividend.
FAC—Divisor value.

Output: FAC—Result of the division.

DATA >0F00 Value Stack Compare (SCOMP)—Compares a value in the VDP RAM stack to the value in FAC.

Input: STACK—VDP RAM address that contains the value to be compared.
FAC—The other value in the comparison.

Output: STATUS—Set according to result. The high bit is set if the value pointed to by STACK is logically higher than FAC. The greater than bit is set if the value pointed to by STACK is arithmetically greater than FAC. The equal bit is set if the values pointed to by STACK and FAC are equal.

DATA >1000 Convert String to Number (CSN)—Coverts an ASCII string to a floating-point number.

Input: FAC+12—Address of the string in VDP RAM.

Output: FAC—Result of the conversion in floating-point format.

DATA >1200 Convert Floating Point to Integer (CFI)—Converts a floating-point number to an integer.

Input: FAC—Floating-point number to be converted.

Output: FAC—The one-word integer value. The maximum value is >FFFF. If an overflow occurs, FAC+10 (>8354) is set to the overflow error code, >03.

DATA >1700 Push Value onto Value Stack (VPUSHG)—Pushes a value from FAC onto the value stack.

DATA >1800 Pop Value from Value Stack (VPOP)—Pops a value from the value stack and places it in FAC.

DATA >2300 Convert Integer to Floating Point (CIF)—Converts an integer to a floating-point number.
Input: FAC—The one-word integer value to be converted.
Output: FAC—Floating-point result.

Linking to Device Service Routines—DSRLNK

Format: BLWP @ DSRLNK — Equates DSRLNK to >6038.
DATA *console-routine-code* — Defines code of DSR routine to be executed.

DSRLNK links an assembly language program to any Device Service Routine (DSR) or subprogram in ROM. The data given is >8 for linkage to a Device Service Routine and >10 for linkage to a subprogram. Before this routine is called, a Peripheral Access Block (PAB) must be set up in VDP RAM. A PAB is a block of memory that contains information about the file to be accessed. In addition, CPU RAM addresses >8356 through >8357 must contain a pointer to the device or subprogram name length in the PAB.

After the routine is executed, information is passed back to your assembly language program in the UTLTAB area. For example, suppose that the following instructions are executed.

```
REF     DSRLNK
 .
 .
 .
BLWP    @DSRLNK
DATA    >8
```

If no error occurs, the equal bit in the Status Register is reset on return from DSRLNK. If an I/O error occurs, the equal bit is set, and the error code is stored in the most significant byte of Register 0 of the calling program's workspace.

If calling the RS232 Device Service Routine, your program must preserve and then restore the values stored in the GROM-Read and GROM-Write addresses. The following program segment shows how to adjust these values.

```
        REF  GRMRA
        REF  GRMWA
              .
              .
              .
SAVEG   BSS  2
              .
              .
              .
        MOVB @GRMRA,@SAVEG
        MOVB @GRMRA,@SAVEG+1
        DEC  @SAVEG
              .
              .
              .
        BLWP @DRSLNK
        DATA >8
              .
              .
              .
        MOVB @SAVEG,@GRMWA
        MOVB @SAVEG+1,@GRMWA
```

Note: Since the cassette DSR is in the GROM, it must be accessed through GPLLNK, rather than DSRLNK. To access a cassette, use BLWP @GPLLNK with DATA >003D.

TI BASIC INTERFACE UTILITIES

These utilities allow an assembly language program to read or assign values to variables passed in a parameter list from a CALL LINK statement in a TI BASIC program. These utility routines include argument-passing utilities and an error-reporting utility.

All of the argument-passing routines use their own workspace area, located at >7092. However, all the parameters are passed through the calling program's workspace.

The following sections describe the data-passing conventions and the calling-statement syntax for each routine.

Numeric Assignment — NUMASG

Format: BLWP @NUMASG Equates NUMASG to >6040.

This routine assigns a numeric value to a numeric variable passed as an argument.

R0	Zero if a simple numeric variable is used, or an array element number if an assignment is made to an array element. The assignment utility tests for legal bounds on the element number. With OPTION BASE 0, the element number must range from 0 to (maximum number of elements – 1). With OPTION BASE 1, the element number must range from 1 to the maximum number of elements.
R1	Argument number (full word) as it appears in the argument list of the CALL LINK statement.
>834A	FAC area. Contains an 8-byte floating-point value to be assigned to the variable.

If the requested argument is not a numeric variable or a numeric array element, an error message is issued.

String Assignment — STRASG

Format: BLWP @STRASG Equates STRASG to >6048.

This routine assigns a string to a string variable passed as an argument to your assembly language program. The utility does the following.

- Allocates space for the string in VDP RAM.
- Copies the string into the allocated VDP RAM.
- Assigns the string to the selected variable.
- Modifies the original argument stack entry to point to the new string. The string to be assigned must be created in RAM by your assembly language program. The first byte of the string contains the length of the string.

The registers are assigned the following values.

R0	Zero if a string is assigned to a simple string variable, or an array element number if assigned to an array element. With OPTION BASE 0, the element number must range from 0 to (maximum number of elements – 1). With OPTION BASE 1, the element number must range from 1 to the maximum number of elements.
R1	Argument number as it appears in the argument list of the CALL LINK statement (one full word).
R2	Address of the string to be assigned. The string must be in RAM.

If the argument specified is not a string variable or an element of a string array, an error message is issued.

Get Numeric Parameter — NUMREF

Format: BLWP @NUMREF Equates NUMREF to >6044.

This utility retrieves the value of a numeric parameter.

R0	Array element number if the argument is a numeric array; otherwise, zero.
R1	Parameter number as it appears in the argument list of the CALL LINK statement.
>834A	FAC area. The beginning address of an 8-byte value of the numeric parameter, returned by the utility routine.

Get String Parameter — STRREF

Format: BLWP @STRREF Equates STRREF to >604C.

This routine retrieves the value of a string parameter. Your program must allocate space in RAM memory before calling this routine, and the first byte of this allocated buffer must contain the maximum buffer length. If the string does not fit in the buffer, an error condition occurs.

R0	Array element number if the argument is a string array; otherwise, zero.
R1	Parameter number as it appears in the argument list of the CALL LINK statement.
R2	Address of the buffer you assign.

If the string fits in the buffer, the string is copied into the buffer following the length byte, and the length byte is modified to reflect the actual length of the string.

Error Reporting — ERR

Format: BLWP @ERR Equates ERR to >6050.

This routine transfers control to the error-reporting routine in the TI BASIC interpreter. The assembly language program may report any existing TI BASIC error or warning message upon returning to TI BASIC.

R0	Error code in the most significant byte.

The error messages that can be issued from your program are listed in the following table.

CAUTION

Error codes smaller than >10 are reserved for the Mini Memory module. Therefore, using these codes in your program can cause unpredictable side-effects.

Code	*Error Message*	*Code*	*Error Message*
00	DSR error–Bad Name	14	Number Too Big
01	DSR error–Write Protected	15	String-Number Mismatch
02	DSR error–Bad Attribute	16	Bad Argument
03	DSR error–Illegal Operation	17	Bad Subscript
04	DSR error–Buffer Full	18	Name Conflict
05	DSR error–Read Past EOF	19	Can't Do That
06	DSR error–Device Error	1A	Bad Line Number
07	DSR error–File Error	1B	For-Next Error
08	Memory Full (closes file)	1C	I/O Error (assumes PAB address in >8304)
09	Incorrect Statement (N/A)	1D	File Error
0A	Bad Tag	1E	Input Error
0B	Checksum Error	1F	Data Error
0C	Duplicate Definition	20	Line Too Long
0D	Unresolved References	21	Memory Full (does not close file)
0E	Incorrect Statement (N/A)	22	Syntax Error
0F	Program Not Found	23	Numeric Overflow
10	Incorrect Statement	24	Unrecognized Character
11	Bad Name	25	String Truncated
12	Can't Continue	26-FF	Unknown Error
13	Bad Value		

SAMPLE PROGRAM

The following program is similar to the DISPLAY...AT routine in TI Extended BASIC. It illustrates the system utilities which interface with the machine resources and the TI BASIC Interpreter.

The program listing follows the conventions required by the Line-by-Line Assembler, which is stored on the cassette tape included with the Mini Memory module. To enter the program via the Editor/Assembler program, you must follow the conventions described in the *Editor/Assembler* owner's manual. For your convenience, the REFs that are applicable to the Editor/Assembler program are given below.

```
*****************************************************************
*   THESE REFS ALLOW PROGRAM ENTRY VIA THE EDITOR/ASSEMBLER
*****************************************************************
              REF VSBW         VDP SINGL BYTE WRITE
              REF VMBR         VDP MULT BYTE READ
              REF VMBW         VDP MULT BYTE WRITE
              REF NUMREF       GET NUMERIC PARAM.
              REF XMLLNK       EXECUTE ROM ROUTINE
              REF STRREF       GET STRING PARAM.
              REF ERR          EXECUTE ERR RPT ROUTINE
         FC   EQU >834A        FLOATING ACCUM
              DEF DISP$
*****************************************************************
*             THESE EQUATES ALLOW PROGRAM ENTRY
*              VIA THE LINE-BY-LINE ASSEMBLER
*****************************************************************
         FC   EQU >834A        FLOATING ACCUM
         SW   EQU >6024        VDP SINGL BYTE WRITE
         MR   EQU >6030        VDP MULT BYTE READ
         MW   EQU >6028        VDP MULT BYTE WRITE
         NR   EQU >6044        GET NUMERIC PARAM.
         SR   EQU >604C        GET STRING PARAM.
         XM   EQU >601C        EXECUTE ROM ROUTINE
         ER   EQU >6050        EXECUTE ERR RPT ROUTINE
*****************************************************************
```



```
7D00              AORG >7D00

7D00        B1    BSS 256         STRING BUFFER
7E00        B2    BSS 32          LINE BUFFER (FOR SCROLLING)
            *********************************************
            *              REGISTER USAGE
            *
            *    R0  -  TEMP VAR VDP ADDR ERR #(MSBY)
            *    R1  -  TEMP VAR VDP DATA(MSBY) CPU BUFF ADDR
            *    R2  -  TEMP VAR BYTE COUNT STR BUFF PTR
            *    R3  -  STRING BUFF PTR
            *    R4  -  VDP SCREEN ADDR OF CHAR
            *    R5  -  COUNT OF BYTES LEFT TO PRINT
            *    R6  -  SCREEN OFFSET CONST FOR BASIC
            *    R7  -  EDGE OF SCREEN LIMIT VARIABLE
            *    R8  -
            *    R9  -  2ND LEVEL LINKAGE
            *    R10  - BASIC RETURN LINKAGE
            *    R11  - SUBROUTINE LINK FOR BL
            *    R12  - CRU BASE ADDR (UNUSED)
            *    R13  - TEMP VARIABLE FOR SCROLL
            *    R14  - TEMP VARIABLE FOR SCROLL
            *    R15  - TEMP VARIABLE FOR SCROLL
            *********************************************
```

```
          ************************************************
          *      MINIMAL IMPLEMENTATION (DISPLAY...AT)
          *
          *     THE SYNTAX FOR THE BASIC STATEMENT IS:
          *        CALL LINK("DISP$",ROW,COL,STRING)
          *
          *     NOTE THAT ONLY STRINGS MAY BE DISPLAYED,
          *     AND ONLY ONE STRING PER CALL STATEMENT
          *     IS ALLOWED.  THERE IS LIMIT CHECKING ON
          *     ROW & COLUMN VALUES.
          ************************************************

7E20 C28B DS     MOV R11,R10      SAVE LINK TO BASIC
7E22 04C0        CLR R0           CLR TO GET VALUE
7E24 0201        LI R1, 1         INITIALIZE PARAM PNTR
7E26 0001
7E28 06A0        BL @GN           GET ROW NUMBER
7E2A 7E6C
7E2C 06A0        BL @LC           CHECK FOR IN LIMITS
7E2E 7E78
7E30 0001        DATA 1           MIN ROW VALUE
7E32 0018        DATA 24          MAX ROW VALUE
7E34 C120        MOV @FC,R4       GET ROW VALUE
7E36 834A
7E38 0604        DEC R4           ADJUST FOR MACH LANG
7E3A 0A54        SLA R4,5         MPY ROW BY 32
7E3C C1C4        MOV R4,R7        MAKE COPY OF ROW ADDR
7E3E 0581        INC R1           INCR PARAM PNTR
7E40 06A0        BL @GN           GET COL NUMBER
7E42 7E6C
7E44 06A0        BL @LC           CHECK LIMITS
7E46 7E78
7E48 0001        DATA 1           MIN COL=1
7E4A 001C        DATA 28          MAX COL=28
7E4C A120        A @FC,R4         ADD IN ROW VAL
7E4E 834A
7E50 0584        INC R4           ADJ FOR BASIC
7E52 0581        INC R1           INCR PARAM PNTR
```



```
7E54 0202        LI R2,B1         LOAD FIRST BUFFR PNTR
7E56 7D00
7E58 0712        SETO *R2         MAX STR LEN = 255
7E5A 0420        BLWP @SR         GET BASIC STRING
7E5C 604C
7E5E 04C5        CLR R5           CLR BYTE COUNT
7E60 C0C2        MOVE R2,R3       COPY BUFFR PNTR
7E62 D173        MOVB *R3+,R5     GET LENGTH BYTE
7E64 06C5        SWPB R5          LEFT ADJ BYTE
7E66 06A0        BL @PR
7E68 7E8E
7E6A 045A        B *R10           RETURN BASIC
          ************************************************
          *         GN - GET NUMERIC PARAMETER
          *
          *     GETS THE BASIC NUMERIC PARAMETER,
          *     CONVERTS IT TO INTEGER, AND LEAVES THE
          *     RESULT IN FC (FLOATING ACC >834A)
          ************************************************
7E6C 0420 GN     BLWP @NR         GET BASIC NUMB.
7E6E 6044
7E70 0420        BLWP @XM         GO TO ROM CODE
7E72 601C
7E74 1200        DATA >1200        FLTPT-TO-INTGR
7E76 045B        B *R11           RETURN
```

```
             *************************************************
             *    LC - LIMIT CHECK ON SIGNED INTEGER
             *
             *    CHECKS THE WORD VALUE IN FC AGAINST THE
             *    UPPER AND LOWER LIMITS WHICH FOLLOW THE
             *    BL CALL.  IF THE VALUE IS OUTSIDE
             *    LIMITS, THE ERROR ROUTINE IS CALLED
             *    WITH THE CODE FOR 'BAD VALUE'.
             *    OTHERWISE, CONTROL RETURNS TO THE
             *    CALLING PROGRAM.
             *************************************************
7E78 8EE0    LC     C  @FC,*R11+      CHK LOWER LIM.
7E7A 834A
7E7C 1104           JLT EC            IF LESS, ERROR
7E7E 8EE0           C  @FC,*R11+      CHK UPPER LIM.
7E80 834A
7E82 1501           JGT EC            IF MORE, ERROR
7E84 045B           B *R11            RETURN
7E86 1300    EC     LI R0, >1300      LOAD ERROR CODE
7E88 1300
7E8A 0420           BLWP @ER          GO TO ERR ROUTN
7E8C 6050
```



```
             *************************************************
             *       PRINT STRING FROM CPU BUFFER
             *
             *    SINCE THIS PROGRAM IS CALLED FROM BASIC,
             *    THERE IS A SCREEN BIAS OF >60 (I.E.,
             *    SCREEN VAL=ASCII VAL + >60)
             *
             *    UPON ENTRY TO THIS ROUTINE, THE
             *    FOLLOWING REGISTERS ARE INITIALIZED
             *
             *       R3  -    POINTER TO START OF STRING
             *                IN CPU RAM
             *       R4  -    BEGINNING SCREEN (VDP) ADDR
             *       R5  -    LENGTH OF STRING
             *       R7  -    START-OF-ROW ADDR (ROW*32)
             *************************************************

7E8E C24B    PR     MOV R11, R9     SAVE SUBROUTINE LINKAGE
7E90 0227           AI R7,30        ADD EOL OFFSET
7E92 001E
7E94 0206           LI R6, >6000    ASCII OFF CONST
7E96 6000
7E98 C004    RE     MOV R4,R0       MOV VDP ADDR
7E9A D073           MOVB *R3+,R1    GET LEN BYTE
7E9C B046           AB R6,R1        ADD ASCI OFFST
7E9E 0420           BLWP @SW        WRITE ONE BYTE
7EA0 6024
7EA2 0584           INC R4          POINT TO NEXT
7EA4 0605           DEC R5          DEC CHAR COUNT
7EA6 1601           JNE L1          JUMP IF NOT DONE
7EA8 0459           B *R9           RET TO MAIN PRG
7EAA 81C4    L1     C R4,R7         IS NEXT POSITION OFF THE EDGE
7EAC 1AF5           JL RE           IF OK, JUMP & MOVE BYTE
7EAE 0227           AI R7,32        INCR CHEK LIM
7EB0 0020
7EB2 0224           AI R4,4         ADDR PNTR TO NEXT LINE
7EB4 0004
```

```
7EB6 C007         MOV R7,R0       COPY LINEND LIM
7EB8 0950         SRL R0,5        DIVIDE ADDR BY 32
7EBA 0280         CI R0,24        IS IT OUT OF LIMITS
7EBC 0018
7EBE 1AEC         JL RE           IF IN LIMITS,
                                  JMP & SET UP ADDR
7EC0 06A0         BL @SC          SCROLL THE SCREEN
7EC2 7ECE
7EC4 0227         AI R7,-32       BRING LIMIT ON SCREEN
7EC6 FFE0
7EC8 0224         AI R4,-32       BRING ADDR ON SCREEN
7ECA FFE0
7ECC 10E5         JMP RE          SET UP NEW ADDR
            ***********************************************
            *       SCROLL - SCROLL THE SCREEN UP AND FILL
            *                THE BOTTOM LINE WITH SPACES
            ***********************************************
7ECE 0200   SC    LI R0,-32        SET UP SCREEN
7ED0 FFE0
7ED2 0201         LI R1,B2         SET UP BUFFR PNTR
7ED4 7E00
7ED6 0202         LI R2,32         SET UP BUFFR LEN
7ED8 0020
7EDA 0220   L4    AI R0,64         MOV DOWN ONE LINE
7EDC 0040
7EDE 0420         BLWP @MR         READ A LINE INTO BUFFR
7EE0 6030
7EE2 0220         AI R0,-32        ADJUST ADDR FOR ONE LINE UP
7EE4 FFE0
7EE6 0280         CI R0,>2E0       IS THIS LAST
7EE8 02E0
```



```
7EEA 1109         JLT NP          IF NOT, STORE LINE
7EEC 1301         JEQ S1          JUMP IF LAST LINE
7EEE 045B         B *R11          SCROLL IS DONE
7EF0 C341   S1    MOV R1,R13      COPY BUFFR PNTR
7EF2 C382         MOV R2,R14      COPY BUFFER LEN
7EF4 020F         LI R15,>2020    LOAD 2 SPACE DATA
7EF6 2020
7EF8 CF4F   L3    MOV R15,*R13+   MOV ONE BYTE
7EFA 064E         DECT R14        DEC BYTE COUNT
7EFC 16FD         JNE L3          PAD NEXT WORD
7EFE 0420   NP    BLWP @MW        MULT BYTE WRITE
7F00 6028
7F02 10EB         JMP L4
                  END
```

EASY BUG DEBUGGER

EASY BUG is a useful program development tool with which you can debug your assembly language programs and access the memory input/output (I/O) ports of the computer. With EASY BUG, you can:

- Inspect and, optionally, modify the contents of CPU and VDP memory.
- Display the contents of GROM.
- Execute assembly language programs from EASY BUG.
- Directly access the peripheral devices which are connected to the computer via the TMS9900 microprocessor's serial I/O port, the Communications Register Unit (CRU).
- Save and load programs on casssette.

OPERATION

When the EASY BUG option is selected from the master selection list, the following screen is displayed.

```
    ===COMMAND TYPES ARE===
MXXXX  MODIFY CPU MEMORY
GXXXX  DISPLAY GROM MEMORY
VXXXX  MODIFY VDP MEMORY
EXXXX  EXEC. ASSEMBLY PROGRAM
CXXXX  CRU SINGLE-BIT I/O
SXXXX  SAVE CPU MEMORY TO CS1
       (STARTING AT XXXX)
L      LOAD STORAGE FROM CS1
===SPECIAL FUNCTION KEYS ARE===
'AID'   DISPLAY THIS SCREEN
PERIOD  ABORT A COMMAND
ENTER   ENTER COMMAND/DATA
MINUS   DISPLAY LAST MEMORY
        (CURRENT UNCHANGED)
SPACE   DISPLAY NEXT MEMORY
        (CURRENT UNCHANGED)
*NOTE*  CPU RAM 8370-83FF IS
        RESERVED FOR EASY BUG
```

This screen summarizes the commands and special function keys used with EASY BUG. The "X's" following the letter commands indicate a hexadecimal address that you enter.

Press any key except **QUIT** to clear the screen and receive a question mark (?) prompt, asking for a command entry.

COMMANDS AND SPECIAL FUNCTION KEYS

A single letter command is used to execute each routine of EASY BUG. Each command (with the exception of the Load Storage command) should be followed by up to four hexadecimal digits indicating an address. If you enter more than four digits, only the last four are used. If less than four digits are entered, they are treated as the last digits of a four-digit value, with the first digits being zero. After typing a command and an address, press **ENTER** to execute the command.

M (Modify CPU Memory)	Allows you to inspect and, optionally, change the contents of CPU memory.
G (Display GROM Memory)	Allows you to display the contents of GROM memory.
V (Modify VDP Memory)	Allows you to inspect and, optionally, change the contents of VDP memory.
E (Execute Assembly Program)	Allows you to run an assembly language program in CPU RAM.
C (CRU Single-Bit I/O)	Allows you to inspect and, optionally, change individual I/O bits.
S (Save CPU Memory)	Allows you to transfer the contents of CPU memory to an audio cassette.
L (Load Storage)	Allows you to load an assembly language program from cassette into CPU memory.

To stop a command's operation, press the **PERIOD** (.) key. The question-mark prompt reappears.

The **ENTER**, **MINUS**, and **SPACE** function keys are used with the Modify CPU Memory (M), Display GROM Memory (G), Modify VDP Memory (V), and CRU Single-Bit I/O (C) commands. The functions of these keys are included in the descriptions of these commands.

Press **AID** to return to the EASY BUG display screen after the screen has been cleared. This key works only when it is entered immediately after a question-mark prompt from EASY BUG.

Each of the EASY BUG commands is described in the following sections.

Modify CPU Memory — M

Format: Mxxxx (where xxxx is a hexadecimal value)

This command displays the contents of a selected CPU memory location and gives you the option of changing the data in that location. If a memory location is not specified with the command, >0000 is used.

After you type the command and address and press **ENTER**, the specified memory address and its contents are displayed.

To change the contents of the displayed memory address, type a two-digit hexadecimal value and press **ENTER**. The last two digits you type are the value used; thus, if you make a mistake when entering a value, simply keep typing until the last two digits are correct. Notice that the left- and right-arrow keys do not work with EASY BUG.

After a memory location and its contents are displayed, you can press the **SPACE** bar to cause the next location and its contents to be displayed, or the **MINUS** (–) key to display the previous location and its contents.

Notice that if you type a value followed by a **SPACE** or **MINUS**, the content of the memory location is not modified. Only when you press **ENTER** directly after typing a value is the content changed.

Typing a **PERIOD** (.) terminates the command and displays the question-mark prompt.

CPU RAM resides in the console, the Mini Memory module, and the Memory Expansion unit, if attached. It is directly addressable from a TMS9900 assembly language program.

The following example inspects the contents of memory locations >8300, >8301, and >8302; changes the contents of >8302 to >F7; changes the contents of >8303 to >12; and redisplays the contents of >8302 and >8303. Finally, the content of >8304 is inspected but is not changed, since the value entered (>3C) was not followed by pressing **ENTER**. Typing a **PERIOD** terminates the command and returns to the question-mark prompt.

Display	*Entries*
?	M8300 <**ENTER**>
M8300 =00 –>	<**ENTER**>
M8301 =00 –>	<**SPACE**>
M8302 =00 –>	F7 <**ENTER**>
M8303 =00 –>	8A12 <**ENTER**>
M8304 =00 –>	<**MINUS**>
M8303 =12 –>	<**MINUS**>
M8302 =F7 –>	<**SPACE**>
M8303 =12 –>	<**SPACE**>
M8304 =00 –>	3C <**SPACE**>
M8305 =00 –>	<**PERIOD**>
?	

CAUTION

Do not modify the contents of CPU memory addresses >8370 through >83FF since this area of memory is used by EASY BUG.

Modify VDP Memory — V

Format: Vxxxx (where xxxx is a hexadecimal value)

This command displays the contents of a selected VDP memory address and gives you the option of changing the data at that address. If a memory location is not specified, >0000 is used.

Note: Since VDP RAM does not extend beyond >3FFF, this is normally the largest address you enter for the Modify VDP Memory command. If you select a larger address, a value is displayed, but this "phantom" location cannot be altered. Otherwise, this command works like the Modify CPU RAM (M) command.

VDP RAM consists of 16K bytes of memory at addresses >0000 through >3FFF. It normally contains screen-related information used by the Video Display Processor, such as screen image, sprite definition, color tables, and character pattern tables. It is also used, in general, as a storage space by applications programs. In particular, higher memory is used by DSRs (Device Service Routines) to pass I/O information. Application programs also use part of VDP RAM as a buffer for DSRs and as a PAB (Peripheral Access Block) to pass information on a file to the appropriate DSR. See Appendix E for more detailed information on the organization of VDP RAM.

When the TI BASIC language is in use, the VDP RAM also holds the BASIC program, the program symbol table, the value stack, the string space, etc. Do not alter the VDP RAM without sufficient knowledge of the BASIC Interpreter since the interpreter uses the VDP RAM in a special order. A detailed configuration of VDP RAM while TI BASIC is in use is shown in Appendix F.

Since VDP RAM is not directly addressable by the CPU, TMS9900 assembly language code (including instructions and workspace) cannot be executed in VDP RAM.

Display GROM Memory — G

Format: Gxxxx (where xxxx is a hexadecimal value)

This command is used to display the contents of selected GROM memory locations. If a memory location is not specified with the command, >0000 is used.

Since GROM is read-only memory, it is not possible to alter the contents of these locations. Otherwise, this command works like the Modify CPU Memory (M) command.

The computer can address up to eight GROMs. Three GROMs in the console control part of the computer operating system and the TI BASIC Interpreter. Up to five additional GROMs may be located in a Command Module. The number of GROMs in a Command Module depends upon the size of the program in the module.

GROM addresses range from >0000 through >F7FF. Each GROM has 6K bytes of memory, starting from an address with an even-numbered first digit. For example, GROM 0 starts at address >0000 and occupies address space through >17FF; GROM 1 starts at address >2000 and occupies address space through >37FF.

The following is a layout of the GROM memory space.

GROM 0	Locations >0000 through >17FF	Contained in the console
GROM 1	Locations >2000 through >37FF	
GROM 2	Locations >4000 through >57FF	
GROM 3	Locations >6000 through >77FF	Contained in a Command Module
GROM 4	Locations >8000 through >97FF	
GROM 5	Locations >A000 through >B7FF	
GROM 6	Locations >C000 through >D7FF	
GROM 7	Locations >E000 through >F7FF	

Execute Assembly Program — E

Format: Exxxx (where xxxx is a hexadecimal number)

This command is used to run an assembly language program located in CPU RAM.

Program control is passed to the location specified. This address should be an entry point in an assembly language program. If a memory location is not specified with the command, >0000 is used.

CRU Single-Bit I/O — C

Format: Cxxxx (where xxxx is a hexadecimal number)

This command is used to display and, optionally, change the CRU bit at the specified location. If a location is not specified with the command, >0000 is used.

After you type the command and address and press **ENTER**, the specified address is displayed, along with the state of the bit at that location (either zero or one). The state of the bit is indicated by the least significant digit of the two-digit value. The left digit is zero.

For example, a display of

C0201 =00 —>

indicates the bit at address >0201 is a zero (the least significant digit of the two-digit value is zero); whereas a display of

C0202 =01 —>

indicates that the bit at address >0202 is one.

To change the state of a bit, enter a zero or a one.

Save CPU Memory to CS1 — S

Format: Sxxxx (where xxxx is a hexadecimal value)

This command dumps the contents of CPU memory to cassette unit number 1, starting at the specified memory location. This command is used to save the contents of a program and/or data on a cassette tape so that it can be loaded again later.

If no address is specified, the contents of memory are dumped, starting from >0000.

After you type the starting address and press **ENTER**, the prompt

TO?

is displayed. Enter the address of the last memory location you want to dump to cassette tape.

After you enter this address and press the **ENTER** key, the contents of the memory range are dumped to the cassette tape on cassette unit number (CS1).

Note: To save all of the contents of the Mini Memory module, including references and pointers, enter a starting address of >7000 and an ending address of >7FFF.

Load Storage from CS1—L

Format: L

This command loads a program from a cassette tape in a cassette recorder/player. The program is loaded into the same memory space it occupied when it was saved with the S command (see above).

When the question-mark prompt (?) is on the screen, press **L** to load a program from cassette. The computer prints instructions on the screen to help you through the procedure. Follow the directions as they appear on the screen. (Be sure you have connected the recorder and inserted the appropriate cassette tape into the recorder.)

See the *User's Reference Guide* for additional information on loading cassettes.

APPENDIX A

CPU Memory Map

>0000		0000
	Console ROM	
>1FFF		8191
>2000		8192
	Memory Expansion — 8K-byte segment (Low Memory)	
>3FFF		16383
>4000		16384
	Peripheral ROMs (mapped) for device service routines	
>5FFF		24575
>6000		24576
	Mini Memory — 4K-byte ROM segment	
>6FFF		28671
>7000		28672
	Mini Memory — 4K-byte RAM segment (Medium Memory)	
>7FFF		32767
>8000		−32768
	Memory Mapped Devices for VDP, GROM, Sound and Speech CPU RAM at >8300 – >83FF	
>9FFF		−24577
>A000		−24576
	Memory Expansion — 24K-byte segment (High Memory)	
>FFFF		−1

APPENDIX B

Mini Memory ROM Organization

>6000		Standard ROM/GROM Header
>6010	XML >70	NAMLNK — Name Link Routine
>6012	XML >71	TGOBLD — Tagged Object Loader
>6014	XML >72	CIF — Convert Integer to Floating
>6016		Unused
>6018	BLWP @GPLLNK	Link to GROM Routine
>601C	BLWP @XMLLNK	Link to ROM Routine
>6020	BLWP @KSCAN	Keyboard Scan
>6024	BLWP @VSBW	VDP Single Byte Write
>6028	BLWP @VMBW	VDP Multiple Byte Write
>602C	BLWP @VSBR	VDP Single Byte Read
>6030	BLWP @VMBR	VDP Multiple Byte Read
>6034	BLWP @VWTR	VDP Write to Register
>6038	BLWP @DSRLNK	Link to Device Service Routine
>603C	BLWP @LOADER	Tagged Object Loader
>6040	BLWP @NUMASG	Numeric Assignment Routine
>6044	BLWP @NUMREF	Get Numeric Parameter
>6048	BLWP @STRASG	String Assignment Routine
>604C	BLWP @STRREF	Get String Parameter
>6050	BLWP @ERR	Error Reporting Routine
>6054		Start of ROM program Area
	...	
>6F38		Start of Pre-Defined REF/DEF Table
	...	
>6FFF		End of Pre-Defined REF/DEF Table

APPENDIX C

RAM Organization—TI BASIC Files

MINIMEM (the 4K-byte segment in the Mini Memory module)	28672	>7000	ID Word >5AA5
	28674	>7002	File Information—Status Information
	28675	>7003	Logical Record Length
	28676	>7004	End-of-File Pointer
	28678	>7006	Current File Entry Point
	28680	>7008	Start of File Space
		...	
	32767	>7FFF	End of File Space
EXPMEM2 (the 24K-byte segment in the Memory Expansion unit)	−24576	>A000	ID Word >5AA5
	−24574	>A002	File Information—Status Information
	−24573	>A003	Logical Record Length
	−24572	>A004	End-of-File Pointer
	−24570	>A006	Current File Entry Point
	−24568	>A008	Start of File Space
		...	
	−1	>FFFF	End of File Space

APPENDIX D

Mini Memory RAM Organization—Assembly Language Storage

28672	>7000	ID Word >A55A	
28674	>7002	Identifiers for Arguments	
	>7012		
28700	>701C	First Free Address in Medium Memory (>7000->7FFF)	
28702	>701E	Last Free Address in Medium Memory	
28704	>7020	Default Entry Address (>0000)	
28706	>7022	First Free Address in High Memory (>A000->FFE0)	
28708	>7024	Last Free Address in High Memory	
28710	>7026	First Free Address in Low Memory (>2000->3FFF)	
28712	>7028	Last Free Address in Low Memory	
28714	>702A	Checksum Value	Used by Tagged Object Loader
28716	>702C	Pointer to Flag Byte in PAB	
28718	>702E	GPL Return Address	
28720	>7030	CRU Address of Peripheral	
28724	>7034	Device Name Length	
28726	>7036	Pointer to Device Name in PAB	
28728	>7038	Version Number of DSR	
		...	
28730	>703A	80-byte Record Buffer for Loader	
		...	
28810	>708A	NAME Buffer	
		...	
28818	>7092	UTILWS Utility Workspace	
		...	
28824	>7098	DSR Link Routine Workspace (Overlaps with UTILWS)	
		...	
28856	>70B8	USRWSP User Program Workspace Registers	
		...	
28888	>70D8	Linking Loader Workspace Registers	
		...	
28920	>70F8	Internal Data Storage	
		...	
28952	>7118	Free Space	
		...	
32767	>7FFF	Start of User Defined REF/DEF Tables	

APPENDIX E

VDP RAM Memory Map

Address	Contents	Decimal
>0000 – >02FF	Pattern Name Table (>0300 bytes)	0000 – 767
>0300 – >037F	Sprite Attribute List	768 – 895
>0380 – >03FF	Pattern Color Table (>0380 – >3FFF) and Free Space	896 – 1023
>0400 – >077F	Sprite Descriptor Blocks	1024 – 1919
>0780 – >07FF	Sprite Velocity Table	1920 – 2047
>0800 – >0FFF	Pattern Generator Area Default Characters >0900 – >0AFF Also used for PAB Area	2048 – 4095
>1000 – >137F	Free Memory Space Used also for PABs and Buffers	4096 – 4991
>1380 – >34FF	Used as Buffer for Program File Load	4992 – 13567
>3500 – >3FFF	Blocks Reserved for Disk DSR	13568 – 16383

APPENDIX F

VDP RAM with BASIC Interpreter

>0000		0
	Screen	
>02FF		767
>0300		768
	Color and Sprite Table	
>031F		799
>0320		800
	Crunch Buffer	
>03BD		957
>03BE		958
	BASIC Temporaries and Interpreter Roll-Out Area	
>03FF		1023
>0400		1024
	Character Tables	
>05FF		1535
>0600		1536
	Value Stack	
	String Space	
	Dynamic Symbol Table and PABs	
	Static Symbol Table	
	Line Number Table	
	Crunched Program	
>37FF		16383

INDEX

SERVICE AND WARRANTY INFORMATION

These modules are durable devices, but they should be handled with the same care you would give any other piece of electronic equipment. Keep the module clean and dry, and don't touch the recessed contacts.

CAUTION:

The contents of a Command Module can be damaged by static electricity discharges.

Static electricity build-ups are more likely to occur when the natural humidity of the air is low (during winter or in areas with dry climates). To avoid damaging the module, just touch any metal object (a doorknob, a desklamp, etc.) before handling the module.

If static electricity is a problem where you live, you may want to buy a special carpet treatment that reduces static build-up. These commercial preparations are usually available from local hardware and office supply stores.

In Case of Difficulty

If the module does not appear to be operating properly, return to the master title screen by pressing **QUIT**. Turn the computer OFF, withdraw the module, align it with the module opening, and reinsert it carefully. Then turn the computer on, and press any key to make the master selection list appear.

If the module is accidentally removed from the slot while the module contents are being used, the computer may behave erratically. To restore the computer to normal operation, turn the computer console off, and wait a few seconds. Then turn the computer on again.

If you have any difficulty with your computer or the Mini Memory module, please contact the dealer from whom you purchased the unit and/or module for service directions, or see the warranty at the back of this book.

Additional information concerning use and service can be found in your *User's Reference Guide.*

Exchange Center Information

If your module requires service, instead of returning it to your dealer or to a service facility for repair or replacement, you may elect to exchange it for a factory-reconditioned module of the same model (or equivalent model specified by TI) by bringing it in person to one of the exchange centers which have been established across the United States. A handling fee will be charged by the exchange center for in-warranty exchanges. Out-of-warranty exchanges will be charged at the rates in effect at the time of the exchange.

To determine if there is an exchange center in your area, look for Texas Instruments Exchange Center in the white pages of your telephone directory, or look under the Calculator and Adding Machine heading in the yellow pages. Please call the exchange center for availability and exchange fee information. Write our Consumer Relations Department for further details and the location of the nearest exchange center.

Battery Information

The battery in the Mini Memory module should remain active for approximately two years with proper care. For best results, store the module only at normal room temperatures. AVOID PROLONGED EXPOSURE OF THE MODULE TO TEMPERATURES ABOVE 100°F, as high temperatures can shorten battery and component life.

When the battery is no longer active, the module will continue to perform properly while inserted in the console with the power on; however, the memory contents will not be retained if the console is turned off.

To check for proper operation of the battery, follow these steps.

1. With the module in place in the console, store a short TI BASIC program in the module memory by means of the SAVE MINIMEM command, and turn the console off.
2. Wait several seconds and turn the console on again. Then select TI BASIC and load the program from the module memory by using the OLD MINIMEM command.
3. LIST the program to be sure that the program has been loaded into console memory.

When the battery is no longer functioning, return the Mini Memory module to a Texas Instruments Service Facility or the Exchange Center nearest you for replacement with a new or reconditioned module (at TI's option). A service fee will be charged for replacement if the module is no longer in warranty.

TEXAS INSTRUMENTS HOME COMPUTER

THREE-MONTH LIMITED WARRANTY
HOME COMPUTER SOFTWARE MODULE
Texas Instruments Incorporated extends this consumer warranty only to the orginal consumer purchaser.

WARRANTY COVERAGE
This warranty covers the electronic and case components of the software module and cassette. These components include all semiconductor chips and devices, plastics, boards, wiring and all other hardware contained in this module and cassette ("the Hardware"). This limited warranty does not extend to the programs contained in the software module and cassette and in the accompanying book materials ("the Programs").

The Hardware is warranted against malfunction due to defective materials or construction. **THIS WARRANTY IS VOID IF THE HARDWARE HAS BEEN DAMAGED BY ACCIDENT, UNREASONABLE USE, NEGLECT, IMPROPER SERVICE OR OTHER CAUSES NOT ARISING OUT OF DEFECTS IN MATERIALS OR WORKMANSHIP.**

WARRANTY DURATION
The Hardware is warranted for a period of three months from the date of the original purchase by the consumer.

WARRANTY DISCLAIMERS
ANY IMPLIED WARRANTIES ARISING OUT OF THIS SALE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE ABOVE THREE-MONTH PERIOD. TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR LOSS OF USE OF THE HARDWARE OR OTHER INCIDENTAL OR CONSEQUENTIAL COSTS, EXPENSES, OR DAMAGES INCURRED BY THE CONSUMER OR ANY OTHER USER.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you in those states.

LEGAL REMEDIES
This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

PERFORMANCE BY TI UNDER WARRANTY
During the above three month warranty period, defective Hardware will be replaced when it is returned postage prepaid to a Texas Instruments Service Facility listed below. The replacement Hardware will be warranted for three months from date of replacement. Other than the postage requirement, no charge will be made for replacement.

TI strongly recommends that you insure the Hardware for value prior to mailing.

TEXAS INSTRUMENTS CONSUMER SERVICE FACILITIES

Texas Instruments Service Facility
P.O. Box 2500
Lubbock, Texas 79408

Geophysical Services Incorporated
41 Shelley Road
Richmond Hill, Ontario, Canada L4C5G4

Consumers in California and Oregon may contact the following Texas Instruments offices for additional assistance or information.

Texas Instruments Consumer Service
831 South Douglas Street
El Segundo, California 90245
(213)973-1803

Texas Instruments Consumer Service
6700 Southwest 105th
Kristin Square, Suite 110
Beaverton, Oregon 97005
(503)643-6758

IMPORTANT NOTICE OF DISCLAIMER REGARDING THE PROGRAMS

The following should be read and understood before purchasing and/or using the software module and cassette.

TI does not warrant that the Programs will be free from error or will meet the specific requirements of the consumer. The consumer assumes complete responsibility for any decision made or actions taken based on information obtained using the Programs. Any statements made concerning the utility of the Programs are not to be construed as express or implied warranties.

TEXAS INSTRUMENTS MAKES NO WARRANTY, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAMS AND MAKES ALL PROGRAMS AVAILABLE SOLELY ON AN "AS IS" BASIS. IN NO EVENT SHALL TEXAS INSTRUMENTS BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THE PROGRAMS AND THE SOLE AND EXCLUSIVE LIABILITY OF TEXAS INSTRUMENTS, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE SOFTWARE MEDIA. MOREOVER, TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE USER OF THE PROGRAMS.

Some states do not allow the exclusion or limitation of implied warranties or consequential damages, so the above limitations or exclusions may not apply to you in those states.

Texas Instruments invented the integrated circuit, the microprocessor, and the microcomputer. Being first is our tradition.

TEXAS INSTRUMENTS
INCORPORATED
DALLAS, TEXAS

Printed in U.S.A.

1037109-58